Android provides a rich application framework for mobile devices



Android 7 应用程序开发教程

掌握Android应用开发技巧，快速进入安卓移动开发殿堂



- 详细讲解Android 7应用开发技巧
- 可作为Android 7应用开发参考指南
- 提供集成开发环境、源代码下载

李 波 编著

清華大学出版社



Android 7

应用程序开发教程

李 波 编著

清华大学出版社
北 京

内 容 简 介

Android 系统是目前最为流行的智能手机操作系统之一，面向 Android 系统的应用开发是目前的技术热点。本书针对 Android SDK 7，结合全新的 Android Studio 开发环境，对 Android 应用编程基础知识进行讲解，易于读者理论联系实践，尽快掌握 Android 系统编程知识。

本书分为 14 章，使用 Java 开发语言，内容主要包括 Android 系统的发展历史、系统架构、应用程序框架、界面开发、网络访问、多媒体应用程序开发、数据存储等。本书每一章都给出实例，使读者进一步巩固所学的知识，提高综合实战能力。

本书既适合熟悉 Java 编程的 Android 初学者和具有一定 Android 编程经验的用户，也可供广大计算机工作者和软件开发者参考。

图书在版编目（CIP）数据

Android 7 应用程序开发教程/李波编著. —北京：清华大学出版社，2019
ISBN 978-7-302-51755-9

Ⅰ. ①A… Ⅱ. ①李… Ⅲ. ①移动终端－应用程序－程序设计－教材 Ⅳ. ①TN929.53

中国版本图书馆 CIP 数据核字（2018）第 271432 号

责任编辑：夏毓彦
封面设计：王　翔
责任校对：闫秀华
责任印制：丛怀宇

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　**邮　　编：**100084
社 总 机：010-62770175　　**邮　　购：**010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn

印 装 者：清华大学印刷厂
经　　销：全国新华书店
开　　本：190mm×260mm　　**印　　张：**30.25　　**字　　数：**774 千字
版　　次：2019 年 2 月第 1 版　　**印　　次：**2019 年 2 月第 1 次印刷
定　　价：89.00 元

产品编号：075968-01

前　　言

自 2007 年 5 月 Android 开源手机平台问世以来，已经经历了 10 多年的发展。这期间，基于 Android 平台的智能手机迅速占领市场，成为当前最受欢迎的手机操作系统之一。随之而来的是基于 Android 操作系统的应用程序需求多元化，Android 开发技术成为市场求职的新宠。

为了帮助国内开发人员快速掌握 Android 开发技术，获取更好的就业机会，笔者基于 Google 公司 2016 年 5 月发布的 Android SDK 7.0（API Level 24）编写了本书，希望能够帮助广大读者在 Android 开发的道路上入门并且获得提高。本书在编写时综合考虑了自学和教学两方面因素。本书不仅适合高校教学，还适合学生自学，同时也适合有一定开发经验的程序员作为参考书使用。

本书内容

本书共分为 14 章，由浅入深地讲解了 Android 开发的各个方面。本书在讲解过程中穿插大量实例，希望借此帮助读者更好地理解 Android 开发的过程，并获得提高。

本书的前 3 章为基础内容，系统地介绍了 Android 系统的诞生和发展的过程、Android 的系统框架、Android 开发环境的搭建以及 Android 应用程序的基本组件，并且着重讲解了 Android 系统中人机交互的基本组件 Activity 的基本知识。

第 4 章讲解了 Android 开发过程中界面开发相关的知识，包括在用户界面设计过程中常用的布局和组件、Android N 的多窗口和通知分组等新特性以及 Android 处理人机交互事件的方法。

第 5 章讲解了 Intent 的基本知识，并利用 Intent 实现了电话和短信应用程序开发功能。

第 6 章主要讲解了 Android 系统下的多媒体开发技术，实现了音频和视频的播放。通过 Service 和 BroadcastReceiver 实现了后台音频播放的相关功能，通过 Android 提供的硬件编程 API 实现了自己的录像和拍照应用程序。

第 7 章讲解了 Android 系统提供的 4 种数据存储方式，分别为 SharedPreferences、文件存储、数据库存储和 ContentProvider。活用这些数据存储方式，实现数据持久化，是应用程序开发过程中不可回避的问题。

第 8 章讲解了网络编程的相关知识，包括 HTTP 编程、Socket 编程、Bluetooth 编程和 WIFI 编程等。

第 9 章解决了利用 Google 提供的 Google Map API 开发自己的位置服务应用的方法。

第 10 章讲解了 Android SDK 提供的绘图 API，包括 2D 绘图和 3D 绘图两个方面。绘图技术是动画制作和游戏开发的重要技术。

第 11 章讲解了 Android 系统应用程序开发的国际化和本地化技术，借助于该技术，可以使开发人员开发的应用程序不需要做任何修改就可以在全球任意地区正常运行。

第 12 章讲解了 Android 7 提供的文本服务，主要介绍如何使用系统提供的剪贴板功能。

第 13 章讲解了 Android 7 的企业应用开发技术，包括设备管理 API、文本语音 API、TV 应用开发和可穿戴技术几部分。

第 14 章讲解了应用程序发布的相关知识，包括应用程序签名的策略、签名文件的生成、如何对应用程序签名以及如何发布到 Google Play Store。正确地发布自己开发的应用程序是利用 Android 技术赚取第一桶金的前提条件。

由于本书篇幅有限，不可能将 Android SDK 7 的相关知识全部讲解，读者可以参阅 Android SDK 文档获取更多信息。

配套示例源代码下载



为了方便读者学习，本书中使用的相关示例源代码可以从下面的地址下载：
https://pan.baidu.com/s/1fTg7gJsqD9_9eWW0MOnbTQ（密码：**tfh2**）
或者扫描右边的二维码下载。

致谢

本书由李波主编，王博、孙宪丽、关颖、杨弘平、曾祥萍、代钦、衣云龙、吕海华、祝世东、夏炎、王玮、王晓强、郭胜龙、林宏刚等也参与了本书的编写，王祥凤、史江萍、李丰鹏、孙士洁参与了本书的整理校对工作。在此，对在本书的编写过程中提供帮助和支持的朋友表示感谢。由于编者水平有限，编写时间仓促，书中难免有疏漏之处，恳请各位读者批评指正。相关指导意见请发送至 introductionandroid@gmail.com，在此编者表示衷心的感谢。

编者

2018 年 10 月

目　　录

第 1 章

Android 系统概述

1.1 智 能 手 机

1.1.1 什么是智能手机

智能手机（Smart Phone）是指“像个人电脑一样具有独立的操作系统，可以由用户自行安装软件、游戏等第三方服务商提供的程序，通过此类程序来不断对手机的功能进行扩充，并可以通过移动通信网络来实现无线网络接入”的这样一类手机的总称。

“智能手机”这个说法主要是针对“功能手机（Feature Phone）”而言的，本身并不意味着这个手机有多“智能”；从另一个角度来讲，所谓的“智能手机”就是一台可以像电脑那样随意安装和卸载应用软件的手机，而“功能手机”则不能。Java 的出现使后来的“功能手机”具备了安装 Java 应用程序的功能，但是 Java 应用程序的操作友好性、运行效率及对系统资源的使用情况都比“智能手机”差了很多。

智能手机具有五大特点：

（1）具备无线接入互联网的能力，即需要支持GSM网络下的GPRS或者CDMA网络的CDMA 1X或3G（WCDMA、CDMA-EVDO、TD-SCDMA）网络，甚至是4G（HSPA+、FDD-LTE、TDD-LTE）网络。

（2）具有 PDA 的功能，包括 PIM（个人信息管理）、日程记事、任务安排、多媒体应用、浏览网页。

（3）具有开放性的操作系统，可以安装更多的应用程序，使智能手机的功能可以得到无限扩展。

（4）人性化，可以根据个人需要扩展机器功能。

（5）功能强大，扩展性强，第三方软件支持多。

智能手机比传统的手机具有更多的综合性处理能力，与传统手机外观和操作方式类似，但是传统手机使用的是生产厂商自行开发的封闭式操作系统，所能实现的功能非常有限，不具备智能手机的扩展性。

1.1.2 智能手机操作系统

智能手机是一种在手机内安装了相应开放式操作系统的手机，随着通信技术的发展，尤其是第三代移动通信技术（3G）的逐步成熟，市场上对功能更强、扩展性能更好的智能手机的需求量增长迅猛。具备独立的操作系统是智能手机最重要的特征。智能手机操作系统是一种运算能力及功能比传统功能手机系统更强的手机系统。智能手机操作系统领域也是各大手机厂商争夺的焦点。目前，主流的智能手机操作系统主要有 Symbian OS、Windows Phone、iOS、Palm OS、BlackBerry OS 和 Android 六种。

各系统的特点如下。

1. Symbian OS

塞班操作系统（Symbian OS）最初是由 Symbian 公司（诺基亚、索尼爱立信、摩托罗拉、西门子等几家大型移动通信设备商共同出资组建的一个合资公司，专门研发手机操作系统）开发的，其前身是 Psion 公司推出的 EPOC（Electronic Piece of Cheese）操作系统，是专门用于智能手机和移动设备的 32 位抢占式、多任务操作系统。其内核与 GUI（Graphical User Interface，图形用户界面，又称图形用户接口）分开，功耗低、占用内存少。

Symbian 操作系统在智能移动终端上拥有强大的应用程序以及通信能力，这都要归功于它有一个非常健全的、核心强大的对象导向系统、企业用标准通信传输协议以及完美的 Sun Java 语言。Symbian 认为无线通信装置除了要提供声音沟通的功能外，同时也应具有其他种类的沟通方式，如触笔、键盘等。在硬件设计上，它可以提供许多不同风格的外形，比如提供真实或虚拟的键盘，在软件功能上可以容纳许多功能，包括和他人分享信息，浏览网页，发送、接收电子邮件和传真，以及个人生活行程管理，等等。此外，Symbian 操作系统在扩展性方面为制造商预留了多种接口，而且 EPOC 操作系统还可以细分成三种类型：Pearl、Quartz 和 Crystal，分别对应普通手机、智能手机和 Hand Held PC 场合的应用。

塞班操作系统为第三方开发商提供一个标准和开放的平台环境。使得第三方应用程序的设计者能够基于该平台开发自己的应用软件。这种方式带来的不足之处是，由于第三方厂商的用户接口程序是不同的，造成了软件不能通用，扩展性较差。这使得塞班操作系统在办公软件和多媒体录放软件上没有开发出足够多的软件供用户使用。

多年来，Symbian 系统一直占据智能系统的市场霸主地位，系统能力和易用性方面均很出色，但是在 Android 系统出现后，Symbian 系统的市场占有率急剧下降。

2. Windows Phone

Windows Phone最早叫Windows Mobile（简称WM），是微软针对移动设备而开发的操作系统。该操作系统的设计初衷是尽量接近桌面版本的Windows，微软按照电脑操作系统的模式来设计

WM，应用软件以Microsoft Win32 API为基础。2010年10月，Windows Phone操作系统正式发布后，Windows Mobile系列正式退出手机系统市场。

微软公司正式发布了智能手机操作系统 Windows Phone，同时将谷歌的 Android 和苹果的 iOS 列为主要竞争对手。2011 年 2 月，诺基亚与微软达成全球战略同盟并深度合作共同研发。2012 年 3 月 21 日，Windows Phone 7.5 登陆中国。6 月 21 日，微软正式发布最新手机操作系统 Windows Phone 8，Windows Phone 8 采用和 Windows 8 相同的内核。

Windows Phone 具有桌面定制、图标拖曳、滑动控制等一系列前卫的操作体验，其主屏幕通过提供类似仪表盘的体验来显示新的电子邮件、短信、未接来电、日历约会等，让人们对重要信息保持时刻更新。它还包括一个增强的触摸屏界面，更方便手指操作，以及一个最新版本的 IE Mobile 浏览器，该浏览器在一项由微软赞助的第三方调查研究中，和参与调研的其他手机浏览器相比，可以执行指定任务的比例高达 48%。很容易看出微软在用户操作体验上所做出的努力，而史蒂夫 • 鲍尔默也表示："全新的 Windows 手机把网络、个人电脑和手机的优势集于一身，让人们可以随时随地享受到想要的体验。"

3. iOS

iOS 在 2011 年 6 月前叫 iPhone OS，是苹果公司为其移动设备开发的操作系统，最初是设计给 iPhone 和 iPod Touch 使用的。与 Mac OS X 操作系统一样，它也是以 Darwin 为基础的。2011 年 6 月之后，iOS 的版本为 5 和 6，通常称为 iOS 5 和 iOS 6。

苹果推出其第一款智能手机 iPhone 后获得了巨大的成功。iOS 继承了 Mac OS X 在个人电脑上界面美观的优势，多点触摸技术的加入为 iPhone 在智能手机领域获得了可观的市场份额。iOS 采用 Quartz 图形框架，能够通过显卡硬件加速实现复杂的图形显示。然而 iOS 是一个不开放的平台，用户不能设计和加载任何第三方的应用程序。这使得 iOS 的扩展性受到很大的限制。

4. Palm OS

Palm OS 是 Palm 公司开发的专用于 PDA 上的一种操作系统，这是 PDA 上的霸主，一度占据了 90%的 PDA 市场的份额。虽然其并不是专门针对手机设计的，但是 Palm OS 的优秀性和对移动设备的支持同样使其能够成为一个优秀的手机操作系统。

Palm 操作系统是多任务的，但每次只允许一个应用程序的打开，多个应用程序不能同时运行，这使得其运行速度很快，具有较好的实用性，但不适应需要多应用程序运行的场合。

5. BlackBerry OS

BlackBerry OS 是 RIM 公司（Research In Motion）专用的操作系统。"黑莓"（BlackBerry）移动邮件设备基于双向寻呼技术。该设备与 RIM 公司的服务器相结合，依赖于特定的服务器软件和终端，兼容现有的无线数据链路，实现了遍及北美、随时随地收发电子邮件的梦想。这种装置并不以奇妙的图片和彩色屏幕夺人耳目，甚至不带发声器。黑莓是目前在美国、加拿大地区相当流行的无线收发电子邮件的软件，它将软件客户端结合在移动电话、PDA 及其他通信终端上，用户可以通过其无线装置来安全地访问电子邮件、企业数据、Web 以及进行企业内部的语音通话。

BlackBerry OS具有多任务处理能力，并支持特定的输入装置，如滚轮、轨迹球、触摸板以及触摸屏等。BlackBerry平台最著名的莫过于它处理邮件的能力。该平台通过MIDP 1.0以及MIDP 2.0的子集，在与BlackBerry Enterprise Server连接时，以无线的方式激活并与Microsoft Exchange、Lotus

Domino或Novell GroupWise同步邮件、任务、日程、备忘录和联系人。该操作系统还支持WAP 1.2。

6. Android

Android 是一种以 Linux 为基础的开放源码操作系统，主要应用于便携设备。Linux 操作系统的嵌入式版本是为各种资源受限的嵌入式终端产品设计的。开放的源码和免费供人使用的特点使得 Linux 的应用开发人员非常丰富。而越来越多的智能手机开发商也倾向于研发 Linux 智能手机，以此来降低手机成本。相比于其他智能手机操作系统，Linux 独有的优势包括以下 4 个方面：

（1）Linux 操作系统几乎能运行在所有主流的处理器上，如 X86、PowerPC、ARM 等。

（2）Linux 作为一个多用户多任务的操作系统，符合 POSIX 便携式计算机环境操作系统接口标准。

（3）Linux 支持和鼓励差异，具有良好的开放性，使得用户可以构筑适合自己的系统。

（4）Linux 是无任何附加条件的开放平台，对硬件平台具有更好的适应性，可移植性强，允许定制用户界面和服务，支持多种格式的可执行文件等。

Android 操作系统最初由 Andy Rubin 开发，最初主要支持手机。2005 年，由 Google 收购注资，并组建开放手机联盟开发改良，逐渐扩展到平板电脑及其他领域。它采用 Linux 2.6.x 版本内核，采用自己的 GUI 架构和应用程序接口，并采用 Java 语言来开发应用程序。它拥有 Linux 操作系统的开放性、对硬件支持好等优点，并且界面美观，这使得它受到市场的普遍欢迎。Android 的主要竞争对手是苹果公司的 iOS 以及 RIM 的 BlackBerry OS。2011 年第一季度，Android 在全球的市场份额首次超过塞班系统，跃居全球第一。

Android 的特点是开放源代码，它的 SDK 开放给任何开发商，所有开发商都可以随意更改界面。

1.2　什么是 Android

1.2.1　Android 的历史

Android 一词最早出现于法国作家利尔亚当（Auguste Villiers de l'Isle-Adam）在 1886 年发表的科幻小说《未来夏娃》（L'ève future）中，将外表像人的机器起名为 Android。

Android 本意指“机器人”，是一个全身绿色的机器人，绿色也是 Android 的标志。Android 最初由现任 Google 工程副总裁安迪•罗宾（Andy Rubin）开发于 2003 年，于 2005 年被 Google 收购。

Android 是基于 Linux 内核的软件平台和操作系统，是 Google 在 2007 年 11 月 5 日公布的手机系统平台，早期由 Google 开发，后由开放手机联盟（Open Handset Alliance）开发。它采用了软件堆层（Software Stack，又名以软件叠层）的架构，主要分为三部分。底层以 Linux 内核工作为基础，只提供基本功能；其他的应用软件则由各公司自行开发，以 Java 作为编写程序的一部分。Android 在未公开之前常被传闻为 Google 电话或 gPhone。大多传闻认为 Google 开发的是自己的手机电话产品，而不是一套软件平台。

1.2.2　Android 的发展

2003 年 10 月，Android 公司在加州 Palo Alto 市成立，联合创始人为 Andy Rubin、Rich Miner、Nick Sear 与 Chris White。

2005 年 8 月，Google 收购了成立仅 22 个月的高科技企业 Android 公司。

2007 年 11 月 5 日，Google 公司正式向外界展示 Android 操作系统。Google 与 34 家手机制造商、软件开发商、电信运营商和芯片制造商共同创建开放手持设备联盟。

2008 年 5 月 28 日，Patrick Brady 于 Google I/O 大会上提出 Android HAL 架构图，8 月 18 日，Android 获得美国联邦通信委员会的批准。

Android 软件一经推出，版本升级非常快，几乎每隔半年就有一个新的版本发布。2008 年 9 月发布 Android 第一版 Android 1.1。后从 Android 1.5 版本开始，Android 用甜点作为它们系统版本代号的命名方法。

2009 年 4 月 30 日，官方 1.5 版本 Cupcake（纸杯蛋糕）正式发布。

2009 年 9 月 15 日，Android 1.6 Donut（甜甜圈）版本发布。

2009 年 10 月 26 日，Android 2.0/2.0.1/2.1 Eclair（松饼）版本发布。

2010年5月20日，Android 2.2/2.2.1 Froyo（冻酸奶）版本发布。

2010年12月7日，Android 2.3 Gingerbread（姜饼）版本发布。

2011年2月2日，　Android 3.0 Honeycomb（蜂巢）版本发布。

2011年5月11日，Android 3.1 Honeycomb（蜂巢）版本发布。

2011 年 7 月 13 日，Android 3.2 Honeycomb（蜂巢）版本发布。

2011 年 10 月 19 日，　Android 4.0 Ice Cream Sandwich（冰激凌三明治）版本在香港正式发布。2011 年 12 月 20 日，谷歌发布了 Android 4.0 操作系统的最新版本 4.0.3，称其对 Android 系统做出了多处改进，并修复了一些缺陷。

2012 年 6 月 28 日，谷歌在 2012 年的 I/O 开发者大会上发布了 Android 4.1 操作系统，Android 4.1 Jelly Bean（果冻豆）是继“冰激凌三明治”之后的下一版 Android 系统。

2012 年 10 月底，Google 在网上以在线的形式发布了全新的 Android 4.2 系统，以及新一代的 Nexus 系列手机 LG Nexus 4 和平板电脑 Nexus 10。Android 4.2 新系统界面改动不大，代号还称为 Jelly Bean，新增了系统全景拍照以及无线同步输出等实用的小功能，并在系统层面做了更多的优化。

2013 年 7 月 25 日，发布 Android 4.3。

2013 年 11 月，Android 4.4 发布，代号为 KitKat。

2014 年 10 月 16 日，发布 Android 7.0 版本，代号为 Nougat，第一次全面支持 ART，并支持平板和可穿戴设备的开发。

2015 年 3 月，Google 发布了 Android 5.1 版本，主要目的是修复 Android 7.0 版本的 Bug，因此其版本号仍然为 Nougat。

2015 年 5 月 8 日，Google 在 Google I/O 2015 大会上发布了 Android 6.0 版本，版本号为 Marshmallow。

2016 年 5 月 18 日，Google 在 Google I/O 2016 大会上发布了 Android 7.0 版本，版本号为 Android Nougat，又称为 Android N。

本书的编写就是基于 Android 7.0 版本进行的。

1.2.3 Android 的优点

Android 的优点主要包括以下 6 项。

1. Android 性价比高

消费者选择产品，价格是必然要考虑的一个因素，iPhone 虽好，但是价格让一般人望而却步。苹果就像是宝马、奔驰，虽然大家都认为它很好，但是一般人消费不起，只有看的份儿。而 Android 如同大众，满大街跑的都是，甚至有一些型号是可以与宝马、奔驰相媲美的。

虽然 Android 平台的手机廉价，但是其性能却一点也不低廉，触摸效果并不比苹果差到哪里去。Android 平台简单实用，无论是功能还是外观设计，都可以与苹果一决高下。在数量众多的 Android 手机中，消费者总是会找到一款满意的 Android 手机取代价格高昂的 iPhone。

2. 应用程序发展迅速

智能手机玩的就是应用，虽然现在 Android 的应用还无法与苹果相竞争，但是随着 Android 的推广与普及，应用程序的数量增长迅速，Android 应用在可预见的未来是有能力与苹果相竞争的。而来自 Android 应用商店最大的优势是，不对应用程序进行严格的审查。在这一点上优于苹果。

3. 智能手机厂家助力

现在，世界上很多智能手机厂家都加入了 Android 阵营，并推出了一系列的 Android 智能机。摩托罗拉、三星、HTC、LG 等厂家都与谷歌建立了 Android 平台技术联盟。厂商加盟的越多，手机终端就会越多，其市场潜力就越大。

4. 运营商鼎力支持

在国内，三大运营商铆足了劲推广 Android 智能机。联通的“0 元购机”、电信的千元 3G、移动的索爱 A8i 定制机都显示了运营商对 Android 智能机的期望。

在美国，T-Mobile USA、Sprint、AT&T 和 Verizon 都推出了 Android 手机。此外，KDDI（日本）、NTTDoCoMo（日本）、TelecomItalia（意大利电信）、T-Mobile（德国）、Telefónica（西班牙）等众多运营商都是 Android 的支持者，有这么多的运营商支持 Android，自然会占据巨大的市场份额。

5. 机型多，硬件配置优

自从 Google 推出 Android 系统以来，各大厂家纷纷推出自己的 Android 平台手机，HTC、索尼爱立信、魅族、摩托罗拉、夏普、LG、三星、联想等都推出了各自的 Android 手机，机型多样，数不胜数。

6. 系统开源利于创新

Android 是开源的，允许第三方修改，这在很大程度上容许厂家根据自己的硬件更改版本，从而能够更好地适应硬件，与之形成良好的结合。开源能够提供更好的安全性能，也给开发人员提供了一个更大的创新空间，从而使 Android 版本升级更快。

1.3　Android 系统架构

图 1.1 是 Android 操作系统的架构，架构包括 4 层，由上到下依次是应用程序层、应用程序框架层、核心类库和 Linux 内核。其中，核心类库中包含系统库及 Android 运行环境。

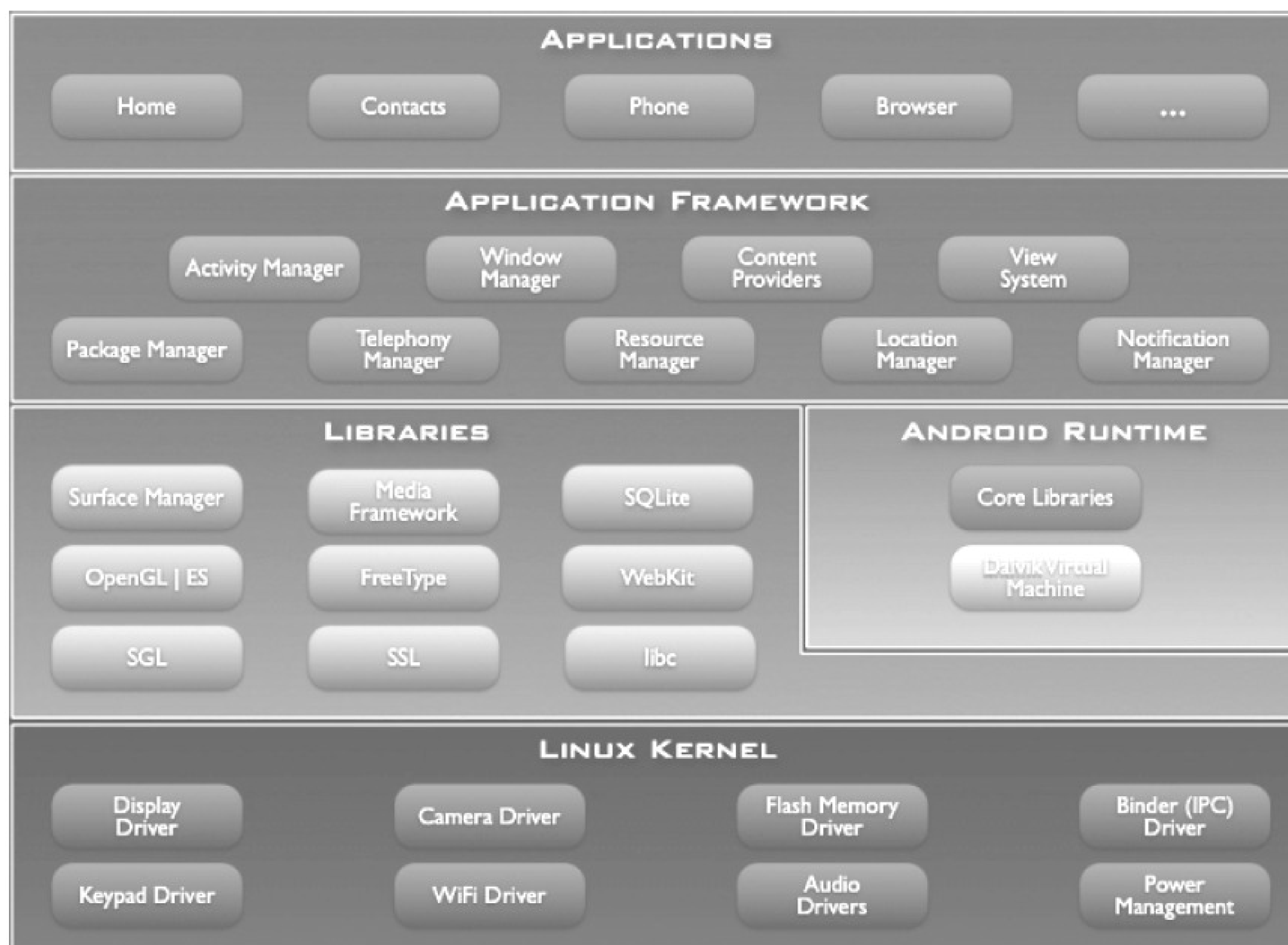


图 1.1　Android 操作系统的架构

1.3.1　应用程序层

Android 装配了一个核心应用程序集合，包括 E-mail 客户端、SMS 短消息程序、日历、地图、浏览器、联系人管理程序和其他程序，所有应用程序都是用 Java 编程语言编写的。用户开发的 Android 应用程序和 Android 的核心应用程序是同一层次的，它们都是基于 Android 的系统 API 构建的。

1.3.2　应用程序框架层

应用程序的体系结构旨在简化组件的重用，任何应用程序都能发布它的功能且任何其他应用程序都可以使用这些功能（需要服从框架执行的安全限制），这一机制允许用户替换组件。开发者完全可以访问核心应用程序所使用的 API 框架。通过提供开放的开发平台，Android 使开发者能够编制极其丰富和新颖的应用程序。开发者可以自由地利用设备硬件优势访问位置信息、运行后台服务、设置闹钟、向状态栏添加通知等。

所有的应用程序都是由一系列的服务和系统组成的，主要包括以下几种：

- 视图（Views）。这里的视图指的是丰富的、可扩展的视图集合，可用于构建一个应用程序，包括列表（Lists）、网格（Grids）、文本框（TextBoxes）、按钮（Buttons），甚至是内嵌的 Web 浏览器。
- 内容管理器（Content Providers）。内容管理器使得应用程序可以访问另一个应用程序的数据（如联系人数据库）或者共享自己的数据。
- 资源管理器（Resource Manager）。资源管理器提供访问非代码资源，如本地字符串、图形和分层文件（layout files）。
- 通知管理器（Notification Manager）。通知管理器使得所有的应用程序都能够在状态栏显示通知信息。
- 活动管理器（Activity Manager）。在大多数情况下，每个 Android 应用程序都运行在自己的 Linux 进程中。当应用程序的某些代码需要运行时，这个进程就被创建并一直运行下去，直到系统认为该进程不再有用为止，然后系统将回收该进程占用的内存以便分配给其他的应用程序。活动管理器管理应用程序生命周期，并且提供通用的导航回退功能。

1.3.3 系统库

Android 本地框架是由 C/C++实现的，包含 C/C++库，以供 Android 系统的各个组件使用。这些功能通过 Android 的应用程序框架为开发者提供服务。

这里只介绍 C/C++库中的一些核心库：

- 系统 C 库。标准 C 系统库（libc）的 BSD 衍生，调整为基于嵌入式 Linux 设备。
- 媒体库。基于 PacketVideo 的 OpenCORE，这些库支持播放和录制许多流行的音频和视频格式，以及静态图像文件，包括 MPEG4、H.264、MP3、AAC、AMR、JPG、PNG。
- 界面管理。管理访问显示子系统，并且为多个应用程序提供 2D 和 3D 图层的无缝融合。
- LibWebCore。新式的 Web 浏览器引擎，支持 Android 浏览器和内嵌的 Web 视图。
- SGL。一个内置的 2D 图形引擎。
- 3D 库。基于 OpenGL ES 1.0 APIs 实现，该库可以使用硬件 3D 加速或包含高度优化的 3D 软件光栅。
- FreeType。位图和矢量字体显示渲染。
- SQLite。SQLite 是一个所有应用程序都可以使用的强大且轻量级的关系数据库引擎。

1.3.4 Android 运行环境

Android 包含一个核心库的集合，该核心库提供了 Java 编程语言核心库的大多数功能。几乎每一个 Android 应用程序都在自己的进程中运行，都拥有一个独立的 Dalvik 虚拟机实例。

Dalvik 是 Google 公司自己设计的用于 Android 平台的 Java 虚拟机。Dalvik 虚拟机是 Google 等厂商合作开发的 Android 移动设备平台的核心组成部分之一。它可以支持已转换为.dex（Dalvik Executable）格式的 Java 应用程序的运行，.dex 格式是专为 Dalvik 设计的一种压缩格式，适合内存和处理器速度有限的系统。Dalvik 经过优化，允许在有限的内存中同时运行多个虚拟机的实例，

并且每一个 Dalvik 应用作为一个独立的 Linux 进程执行。Dalvik 虚拟机依赖 Linux 内核提供基本功能，如线程和底层内存管理。

1.3.5 Linux 内核

Android 基于 Linux 提供核心系统服务，例如安全、内存管理、进程管理、网络堆栈、驱动模型。除了标准的 Linux 内核外，Android 还增加了内核的驱动程序，如 Binder（IPC）驱动、显示驱动、输入设备驱动、音频系统驱动、摄像头驱动、WiFi 驱动、蓝牙驱动、电源管理。

Linux 内核也作为硬件和软件之间的抽象层，它隐藏具体硬件细节而为上层提供统一的服务。分层的好处就是使用下层提供的服务为上层提供统一的服务，屏蔽本层及以下层的差异，当本层及以下层发生了变化时，不会影响到上层，可以说是高内聚、低耦合。

1.4 Android 7 新特性介绍

Android 7.0 Nougat 是迄今为止规模最大的 Android 版本。该版本为用户推出了各种崭新的功能，为开发者提供了数千个新的 API。不仅如此，它还将 Android 扩展得更广，小到手机、平板电脑和穿戴式设备，大到电视和汽车。

本节主要介绍 Android 7 新增的几个特性。

1.4.1 分屏显示

在运行 Android 7 的手机和平板电脑上，用户可以并排运行两个应用，或者处于分屏模式时，一个应用位于另一个应用之上。用户可以通过拖动两个应用之间的分隔线来调整应用所占屏幕的大小，如图 1.2 所示。



图 1.2 分屏显示

1.4.2 全新的通知设计

Android 7 对通知栏功能进行了进一步的丰富，使之速度更快且更加易于使用。可以实现通知栏内容分组、通知样式自定义、通知直接回复等功能，如图 1.3 所示。此外，借助于模板，开发者只需编写少量的代码便可以实现相关功能。

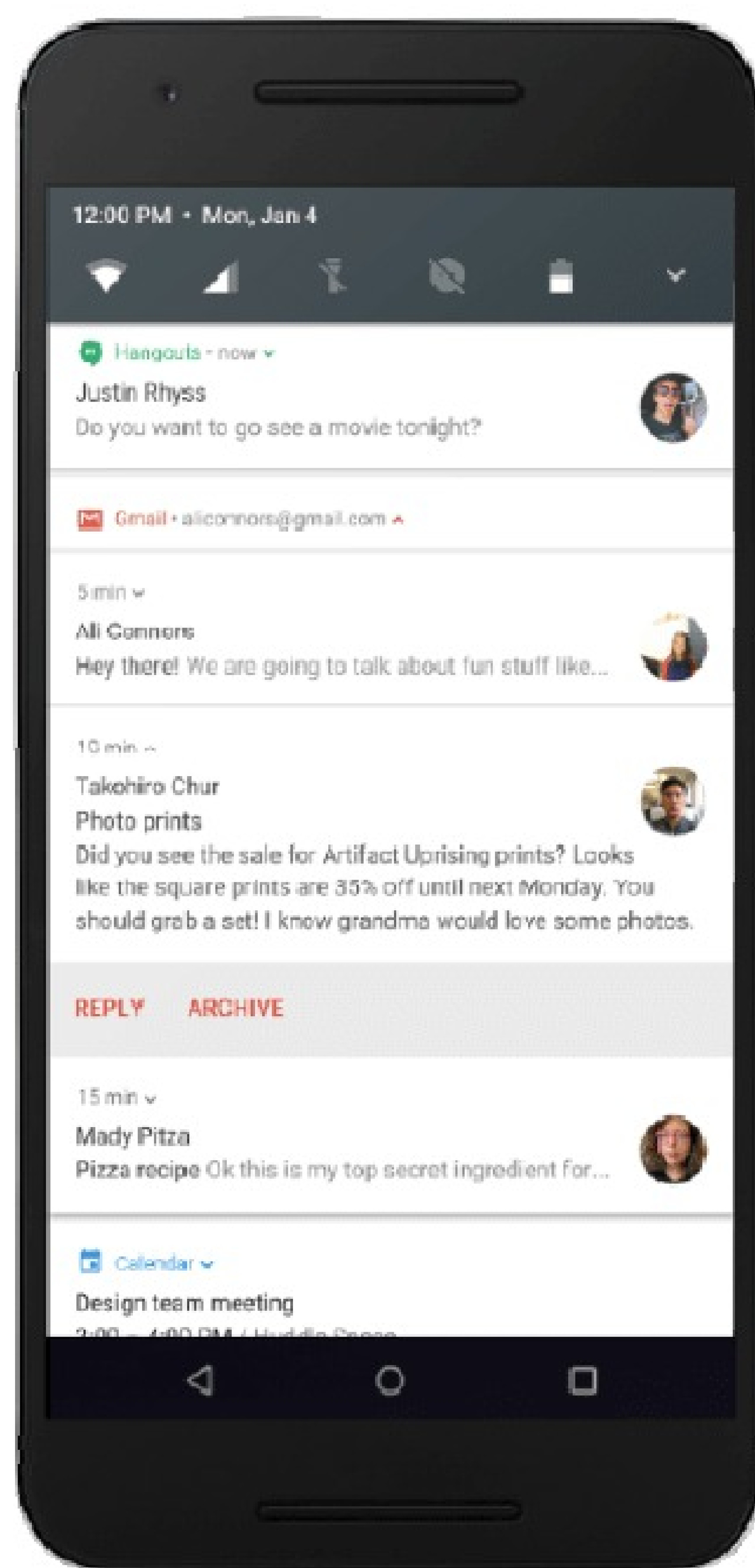


图 1.3　通知直接回复功能

1.4.3　基于配置文件的 JIT/AOT 编译

在 Android N 系统中，添加了 Just in Time（JIT）编译器支持，可以在应用运行时对 ART 进行代码分析，持续提升 Android 应用的性能。JIT 编译器对 Android 运行组件 Ahead of Time（AOT）编译器进行了补充，有助于提升运行时性能，节省存储空间，加快应用更新和系统更新速度。

基于配置文件的 JIT/AOT 编译可以让 Android N 系统的运行组件依据应用的实际情况对应用进行 JIT/AOT 编译，有助于降低 RAM 使用，降低耗电量，并且能够大幅度提升应用的安装速度。

1.4.4　优化的低电耗模式

Android 6.0 推出了低电耗模式，即设备处于空闲状态时，通过推迟应用的 CPU 和网络活动以实现省电目的的系统模式，例如设备放在桌上或抽屉里时。Android N 将低能耗模式更加推进了一步，只要屏幕关闭了一段时间，且设备未插入电源，低电耗模式就会对应用使用熟悉的 CPU 和网络限制。这意味着用户即使将设备放入口袋里也可以省电。

1.4.5　Project Svelte：后台优化

Android N 系统持续改善了 Project Svelte ，以最大程度地减少 Android 设备中一系列系统和应用使用的 RAM。

此外，在 Android N 中，删除了三个常用的隐式广播：CONNECTIVITY_ACTION、ACTION_NEW_PICTURE 和 ACTION_NEW_VIDEO。因为这些广播可能会一次唤醒多个应用的后台进程，同时会耗尽内存和电池。

1.4.6　Data Saver

在移动设备的整个生命周期，蜂窝数据计划的成本通常会超出设备本身的成本。对于许多用户而言，蜂窝数据是他们想要节省的昂贵资源。

Android N 推出了 Data Saver 系统服务（见图 1.4），有助于减少应用使用的蜂窝数据，无论是在漫游、账单周期即将结束，还是使用少量的预付费数据包。Data Saver 让用户可以控制应用使用蜂窝数据的方式，同时让开发者打开 Data Saver 时可以提供更多有效的服务。

此外，Android N 扩展了 ConnectivityManager，以便为应用检索用户的 Data Saver 首选项并监控首选项变更。所有应用均应检查用户是否已启用 Data Saver 并努力限制前台和后台的流量消耗。

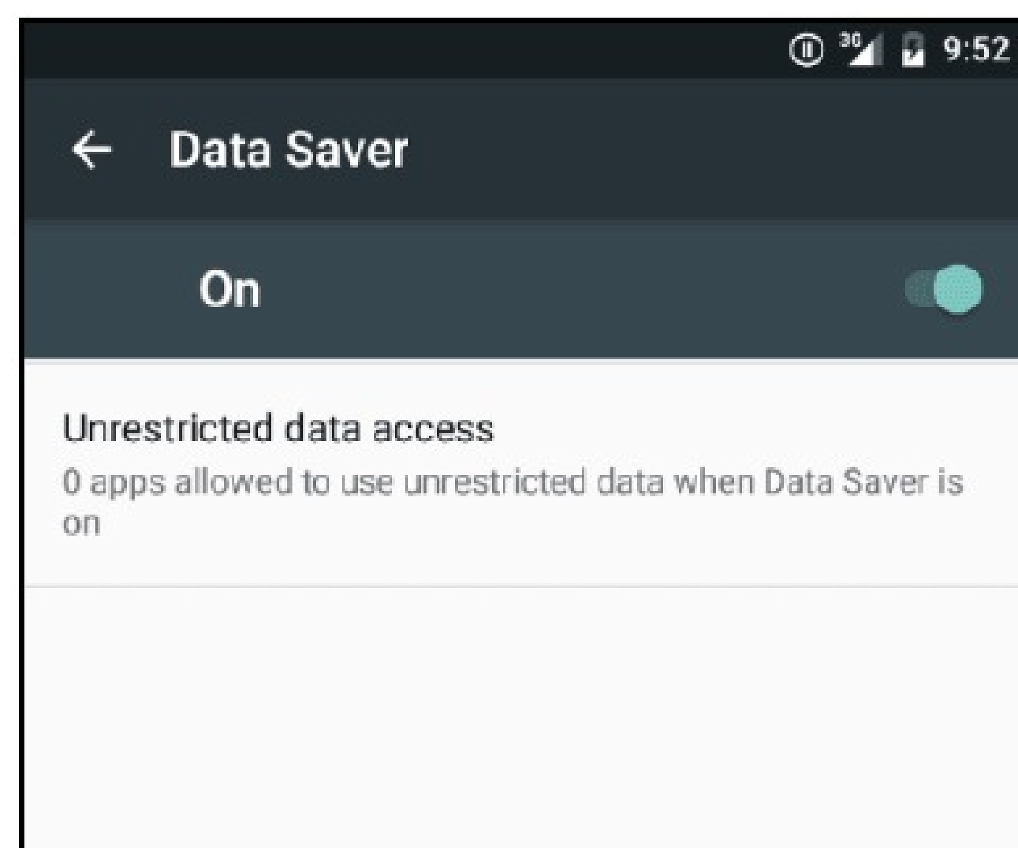


图 1.4　Data Saver

1.4.7　Quick Settings Tile API

快速设置贴片通常用于直接从通知栏显示关键设置和操作，如图 1.5 所示。在 Android N 中，扩展了快速设置贴片的范围，使其使用更加方便。

Android N 为快速设置贴片添加了更多空间，用户可以通过向左或向右滑动跨分页的显示区域访问它们。用户可以控制显示哪些快速设置贴片，并且可以通过拖放贴片来添加或移动贴片位置。

Android N 为开发者提供了新的 API，以定义自己的快速设置贴片，进而使用户能够轻松访问应用中的关键控件和操作。

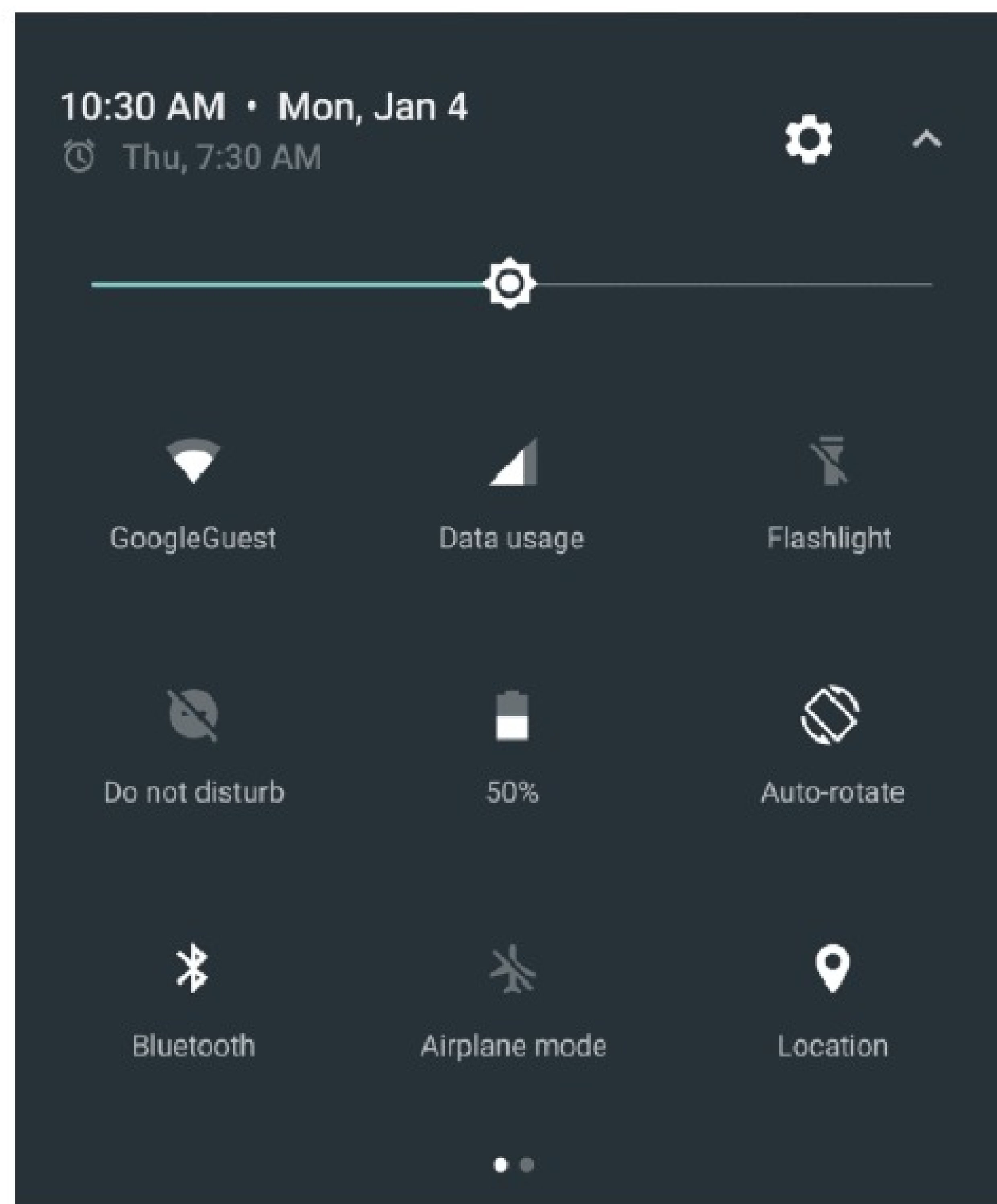


图 1.5　快速设置贴片

1.4.8　号码屏蔽和来电过滤

Android 7.0 对号码屏蔽和来电过滤功能提供了平台级别的支持，并提供了相关的 API。系统会形成一个号码屏蔽列表，系统默认的短信应用、系统应用和服务提供商开发的应用可以访问该列表，而其他应用不具有访问该列表的权限。

来电过滤功能除了会拒绝来电呼入之外，还可以将来电记录到系统日志，并且不向用户发送

来电通知。借助号码屏蔽列表还可以完成短信屏蔽、跨设备使用该列表、多应用共用该列表等功能。

1.4.9 OpenGL ES 3.2 API 支持

Android 7.0 支持 Khronos OpenGL ES 3.1，因此开发者可以在受支持的设备上为游戏和其他应用采用最高性能的 2D 和 3D 图形功能。

OpenGL ES 3.2 增加了计算着色器、模板纹理、加速的视觉效果、优化 ETC2/EAC 纹理压缩、高级纹理渲染、标准化纹理尺寸以及渲染缓冲区格式等功能，针对 HDR 的浮点帧缓冲和延迟着色进行了优化，并通过强大的缓冲区访问控制减少了 WebGL 开销。

Android 7.0 还支持 Android 扩展程序包（AEP），这是一组 OpenGL ES 扩展程序，可让开发者使用镶嵌图案着色器、几何图形着色器、ASTC 纹理压缩、按样本插入和着色以及其他高级渲染功能。有了 AEP，开发者就可以通过一系列 GPU 运用高性能图形。

1.4.10 密钥认证

Android 7.0 使用硬件支持的密钥库，可更安全地在 Android 设备上创建、存储和使用加密密钥。它们可保护 Linux 内核免受潜在的 Android 漏洞的攻击，也可防止别人从已取得 root 权限的设备提取密钥，以此提高 Android N 系统的安全性。

1.5 小　　结

本章介绍了智能手机的概念及其流行的操作系统，并对当前最流行的 Android 操作系统进行了详细介绍，从其产生、发展过程中得出其优势所在。

本章重点介绍了 Android 操作系统的系统构架，从应用程序层、应用程序框架层、核心类库和 Linux 内核 4 部分进行了详细的介绍，并介绍了 Android 7.0 的新特性。

1.6 习　　题

1. 了解 Android 系统的发展过程。
2. Android 系统架构分为哪几层？
3. 系统库中的核心库有哪些？它们的作用分别是什么？

第 2 章

搭建 Android 开发环境

2.1 系统需求

支持 Android 开发的系统如下，读者可以选择自己喜欢的系统平台。

- Windows XP（32 位）、Vista（32 位或 64 位）、Windows 7（32 位或 64 位）、Windows 10（32 位或 64 位）。
- Mac OS X 10.5.8 或以后版本（x86）。
- Linux Ubuntu。

2.2 软件安装

2.2.1 JDK 的安装

JDK 的安装步骤说明如下：

步骤 01 **下载 JDK。通过 Android 系统架构可以知道，要进行开发需要下载并安装 Java 的开发环境。首先需要下载免费 JDK 软件包。Android SDK 需要 JDK 7 以上版本，JDK 包含一整套开发工具。由于 Sun 公司已经被 Oracle 公司收购，因此需要到 Oracle 公司的网站下载，下载地址是：http://www.oracle.com/technetwork/java/javase/ downloads/index.html，值得注意的是，必须下载完整的 JDK 开发包，不可以只安装 JRE 运行版本，下载界面如图 2.1 所示。目前最新版本是 JDK 10，但是**

为了更好的稳定性，建议使用 JDK 8。



图 2.1　Java JDK 下载界面

步骤 02 安装 JDK。双击下载的可执行文件，接受许可后就可以安装了。安装过程比较简单，就不再展开描述了。

步骤 03 配置 Java 环境变量。为了使用 Java 工具进行编译、运行，需要配置 Java 环境变量，采用相对路径的方法，需要设置的三个环境变量：JAVA_HOME、CLASSPATH 和 PATH。假设将 JDK 安装到了 C:\JAVA\JDK8\路径下，则右击“我的电脑” | “属性” | “高级” | “环境变量”。

- 配置 JAVA_HOME：JAVA_HOME= “C:\JAVA\JDK8\”。
- 配置 CLASSPATH：CLASSPATH= “.; %JAVA_HOME%\jre\lib\rt.jar;”。
- 配置 PATH：PATH= “%JAVA_HOME%\bin;”。

2.2.2　Android Studio

开发 Android 应用程序需要下载相关的 Android SDK。到 http://developer.Android.com/sdk/index.html 开发网页，如图 2.2 所示，根据自己的操作系统下载 Android SDK 软件开发包。本书下载的是 Android 7.0 版本（API Level 24）。本书使用官方推荐的 Android Studio 进行开发，版本号是 2.2.3，Gradle 版本是 2.3.3。

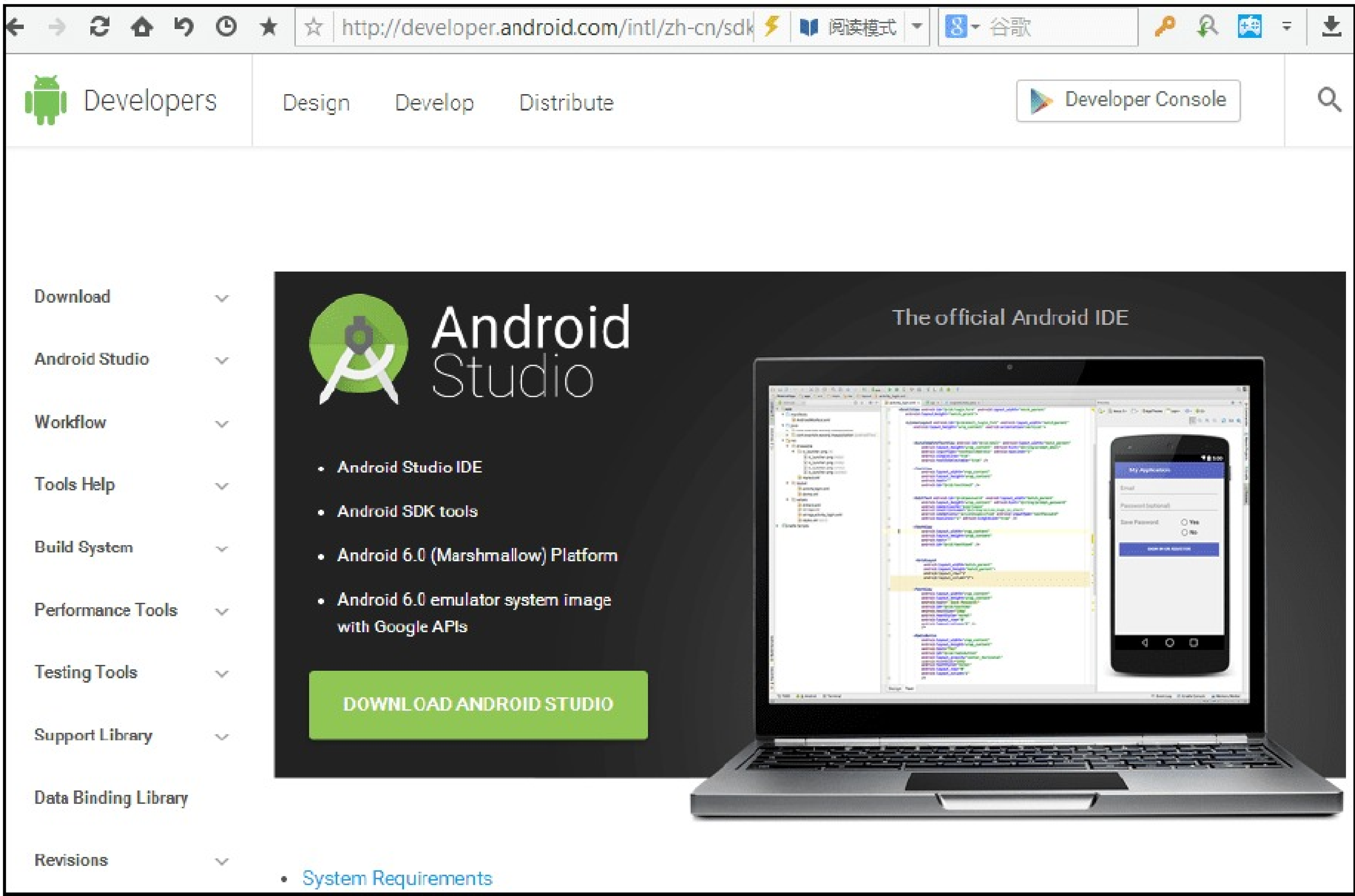


图 2.2　Android SDK 下载页

下载完成后，双击即可安装。Android Studio 包含开发 Android 应用所需要的文件、运行环境及相关工具，如图 2.3 所示。

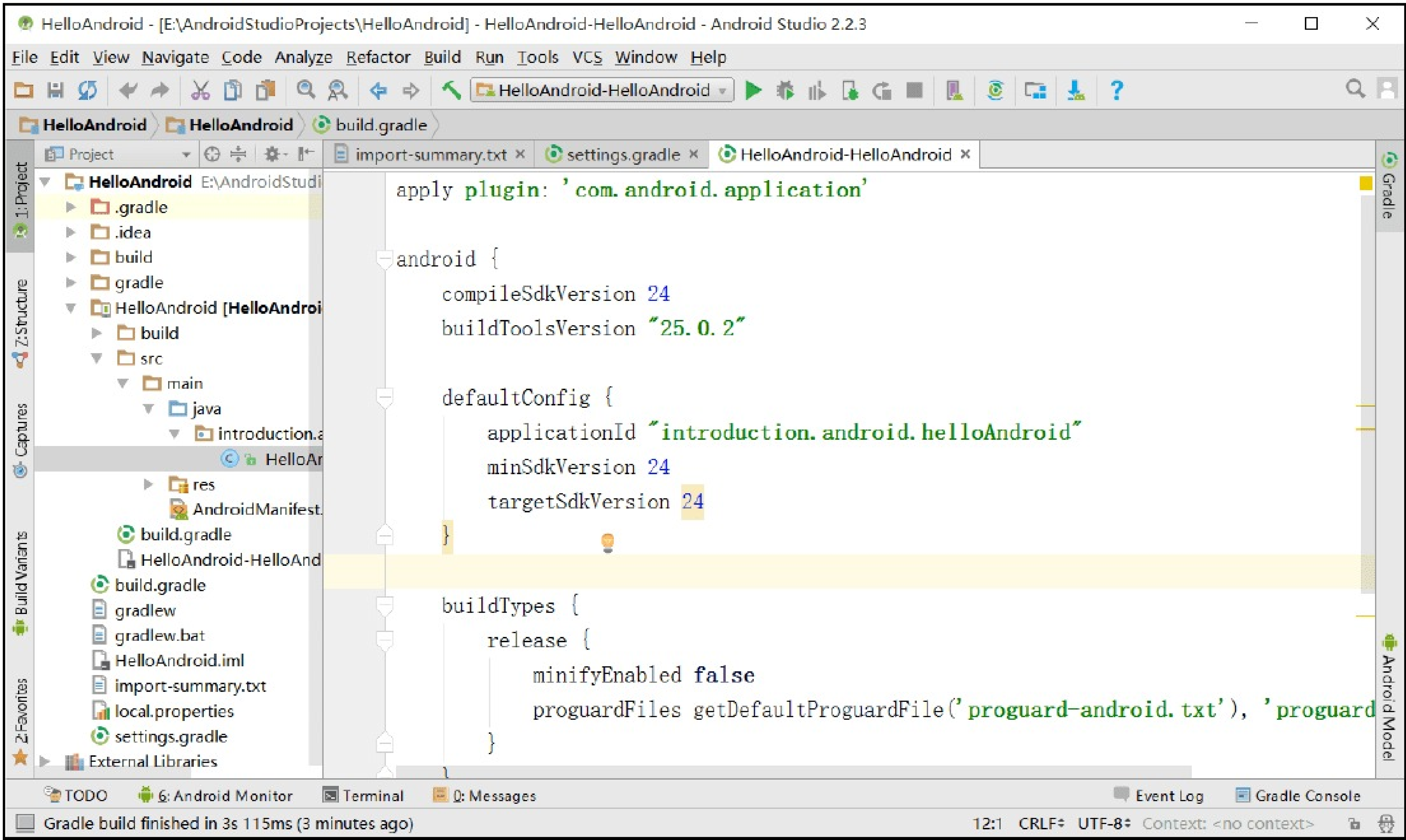


图 2.3　Android Studio 运行界面

Android Studio 的“Tools”菜单下包含一个“Android”菜单项，如图 2.4 所示，单击其中的子菜单“SDK Manager”会启动 SDK 管理器。通过 SDK 管理器可以查看本机已经安装的 Android SDK 版本，如图 2.5 所示。

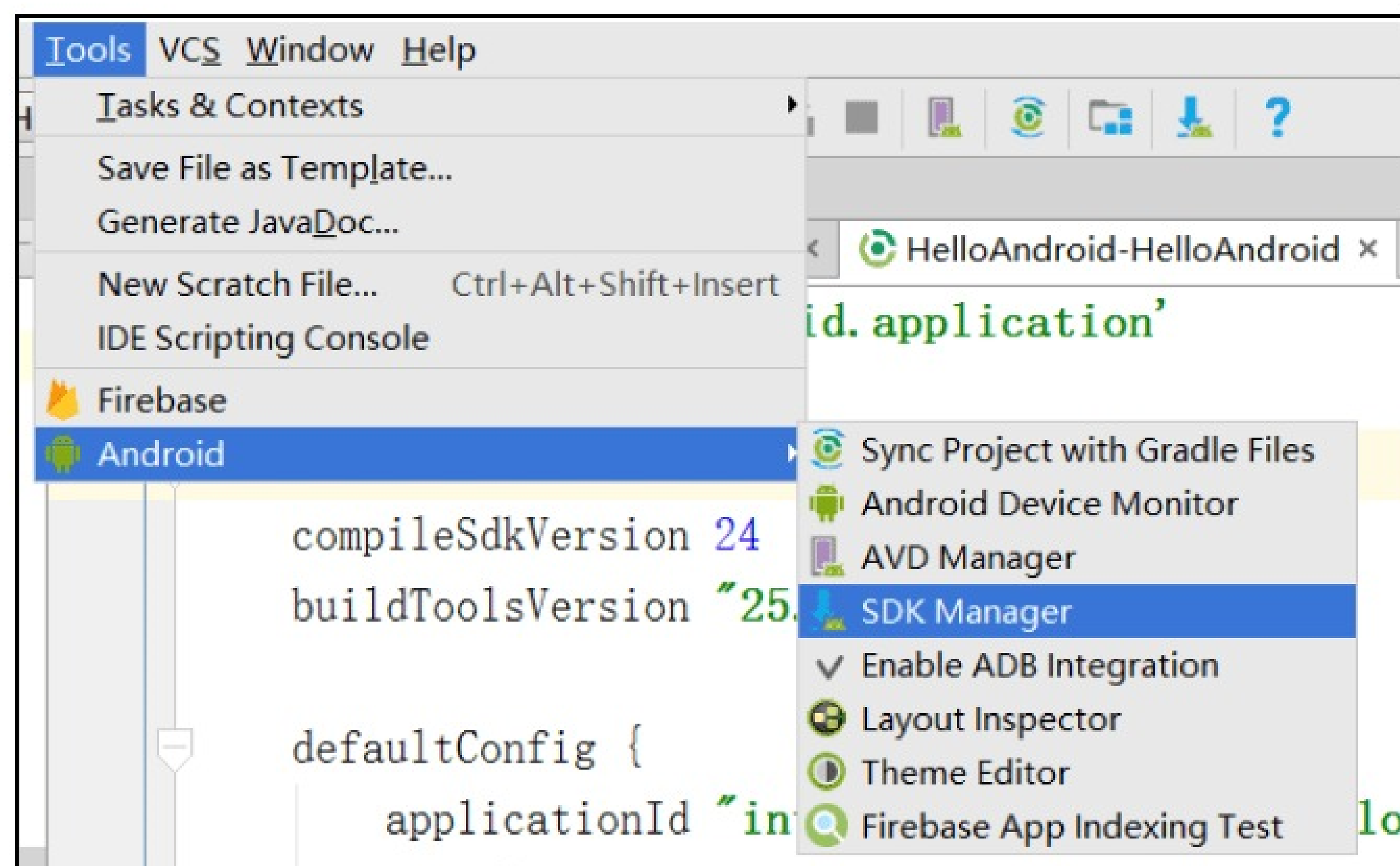


图 2.4　Android 子菜单

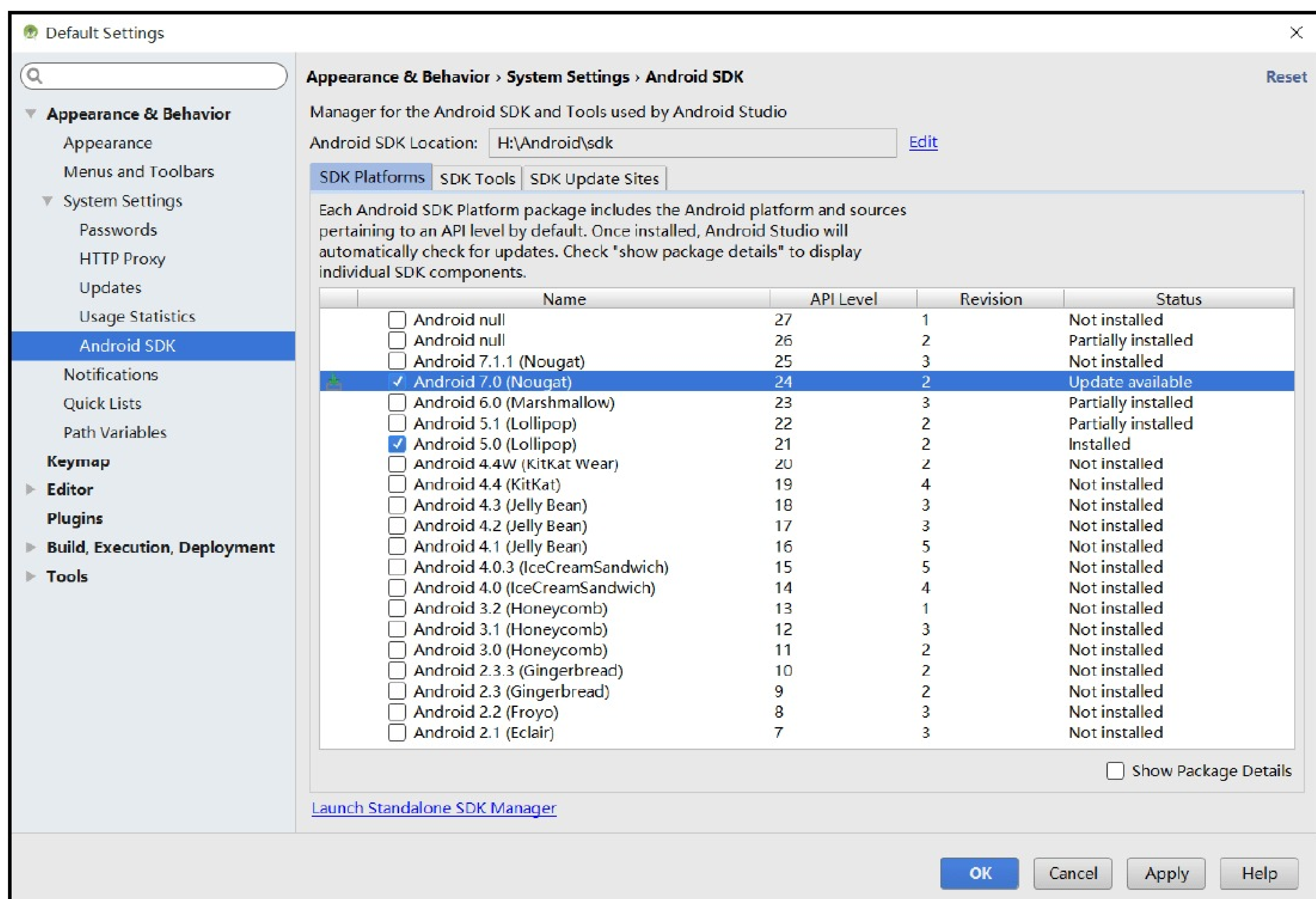


图 2.5　SDK Manager

单击“Launch Standalone SDK Manager”会启动独立的 SDK 管理器，如图 2.6 所示，可完成对 SDK 的文档、工具等进行相应的安装和更新工作。

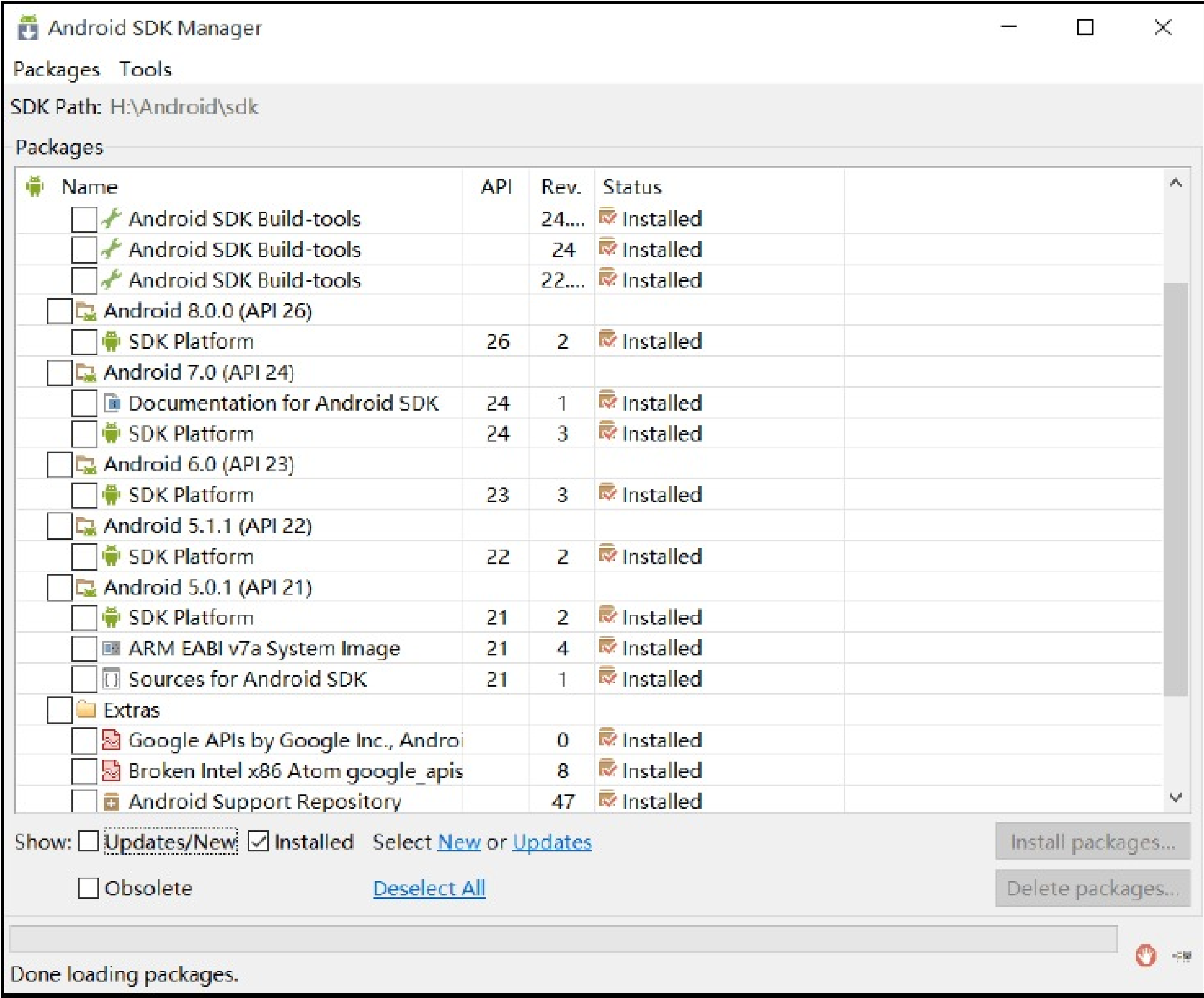


图 2.6　独立的 SDK 管理器

2.2.3　创建 AVD

在 Android Studio 中单击 Tools|Android| AVD Manager 命令，启动 Android 虚拟设备管理器，如图 2.7 所示。单击"Create Virtual Device"按钮，出现新建虚拟设备界面，如图 2.8 所示。总体而言，界面分为左中右三部分，左侧为 TV、Wear、Phone、Tablet 四个类别，说明 Android 7 对电视、可穿戴设备、手机和平板的开发都提供了支持；中间一列为针对左侧的某个类别已经建立好的虚拟设备的配置文件，可基于配置文件直接创建虚拟设备；右侧为配置文件的图形化描述，包括屏幕尺寸、现实精细度等。

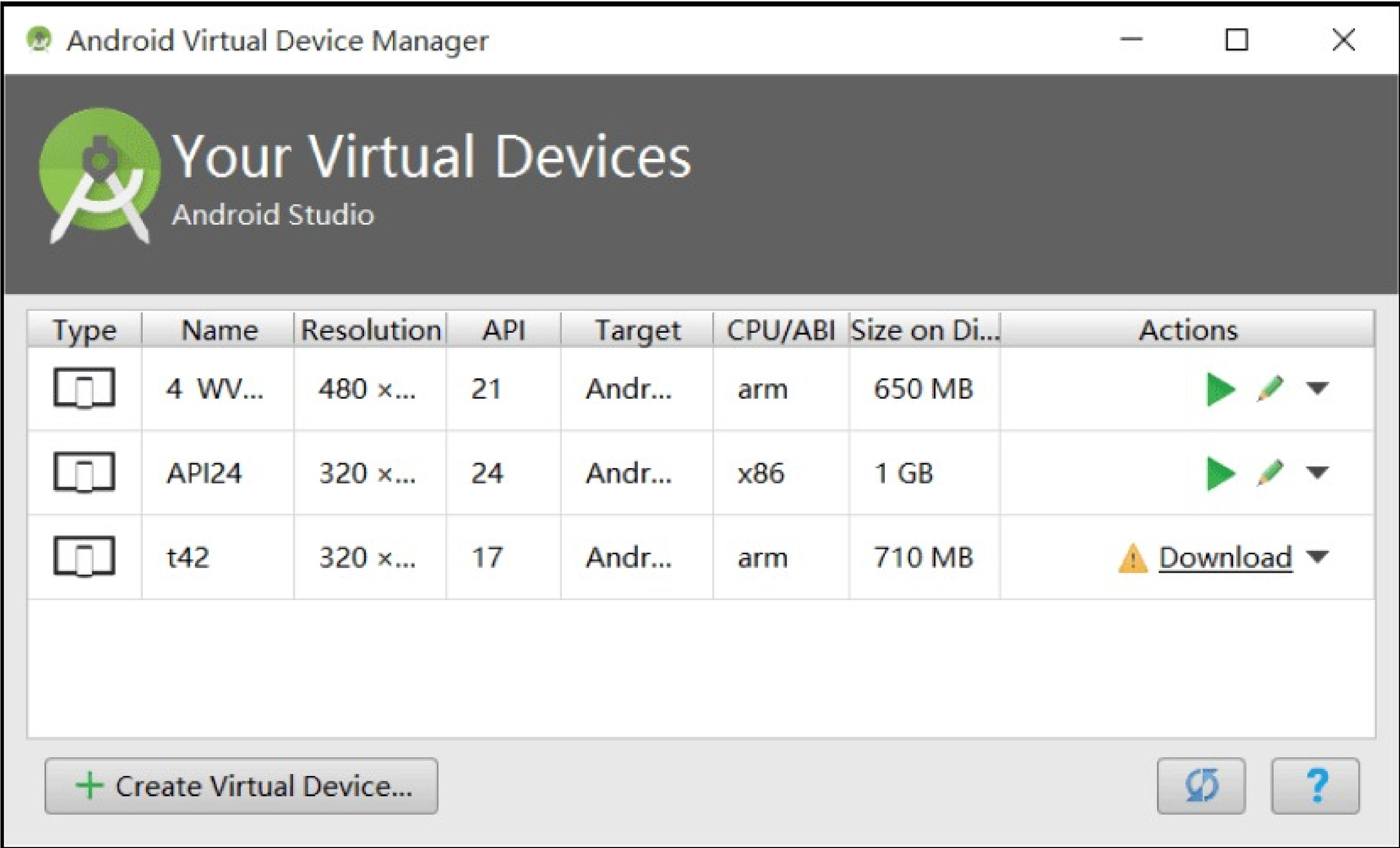


图 2.7　AVD 管理器

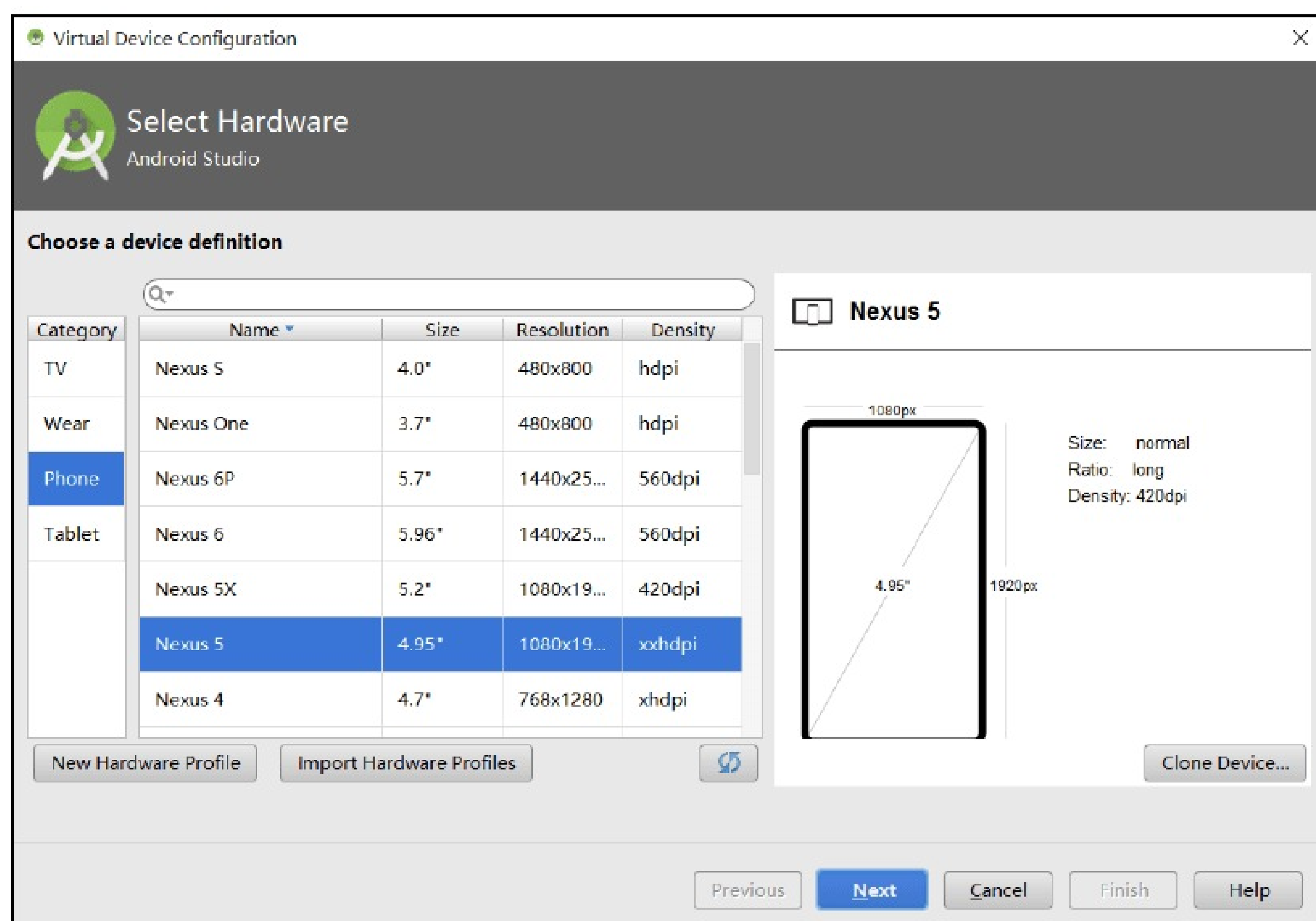


图 2.8　新建虚拟设备界面

例如，要基于 Nexus S 配置文件创建虚拟手机，其分辨率为 480×800，现实效果为 hdpi，需要在左侧单击“Phone”，在中间选择“Nexus S”配置文件，然后单击“Next”按钮，出现系统映像选择界面，如图 2.9 所示，选择系统映像文件，决定虚拟手机的 Android 系统版本、系统架构以及 API 等级。Android 7 支持 x86 架构、x86_64 架构、armeabi 架构以及 arm64 架构，可根据需要进行选择。选择 Nougat，API Level 为 24，架构为 x86，单击“Next”按钮，进入虚拟设备参数配置界面，如图 2.10 所示，为虚拟手机设备起一个名字，并可对虚拟设备的分辨率、Android 系统版本、横屏还是竖屏、3D 绘图使用硬件加速还是软件加速等信息进行配置。最后单击“Finish”按钮，完成虚拟手机设备的创建。创建的虚拟设备会出现在 AVD 管理器中，单击运行即可启动，如图 2.11 所示。

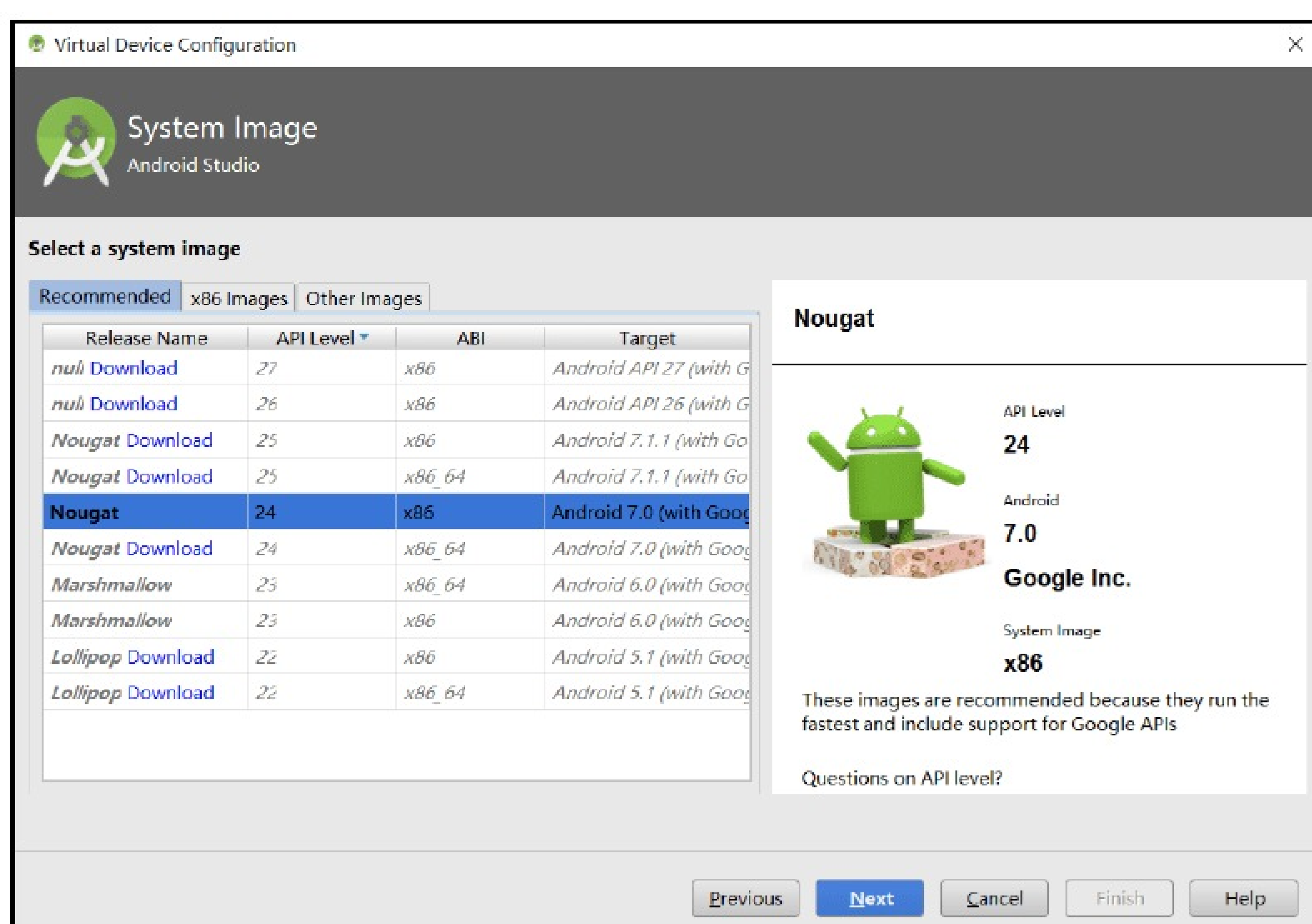


图 2.9　系统映像选择

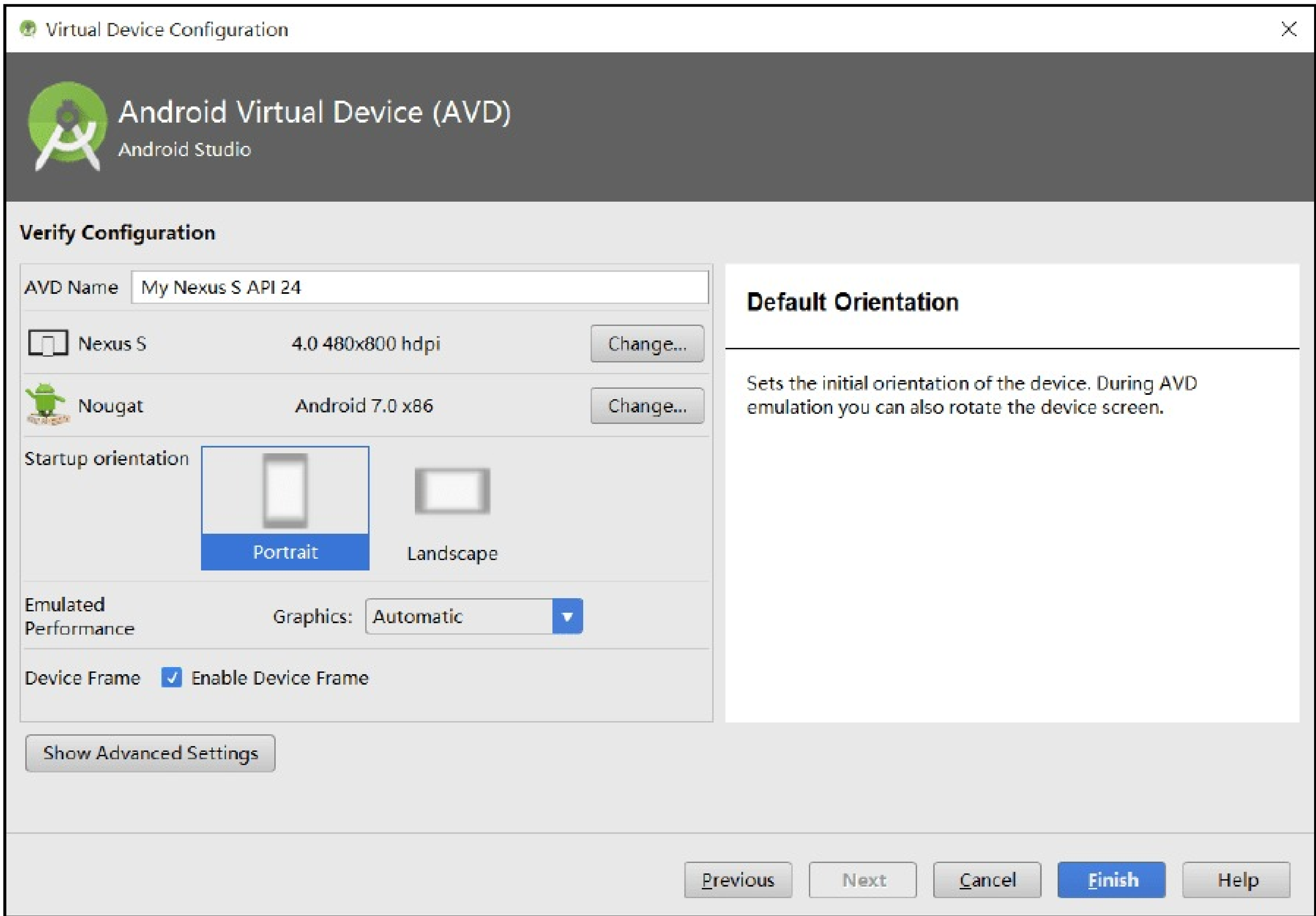


图 2.10　虚拟设备参数配置

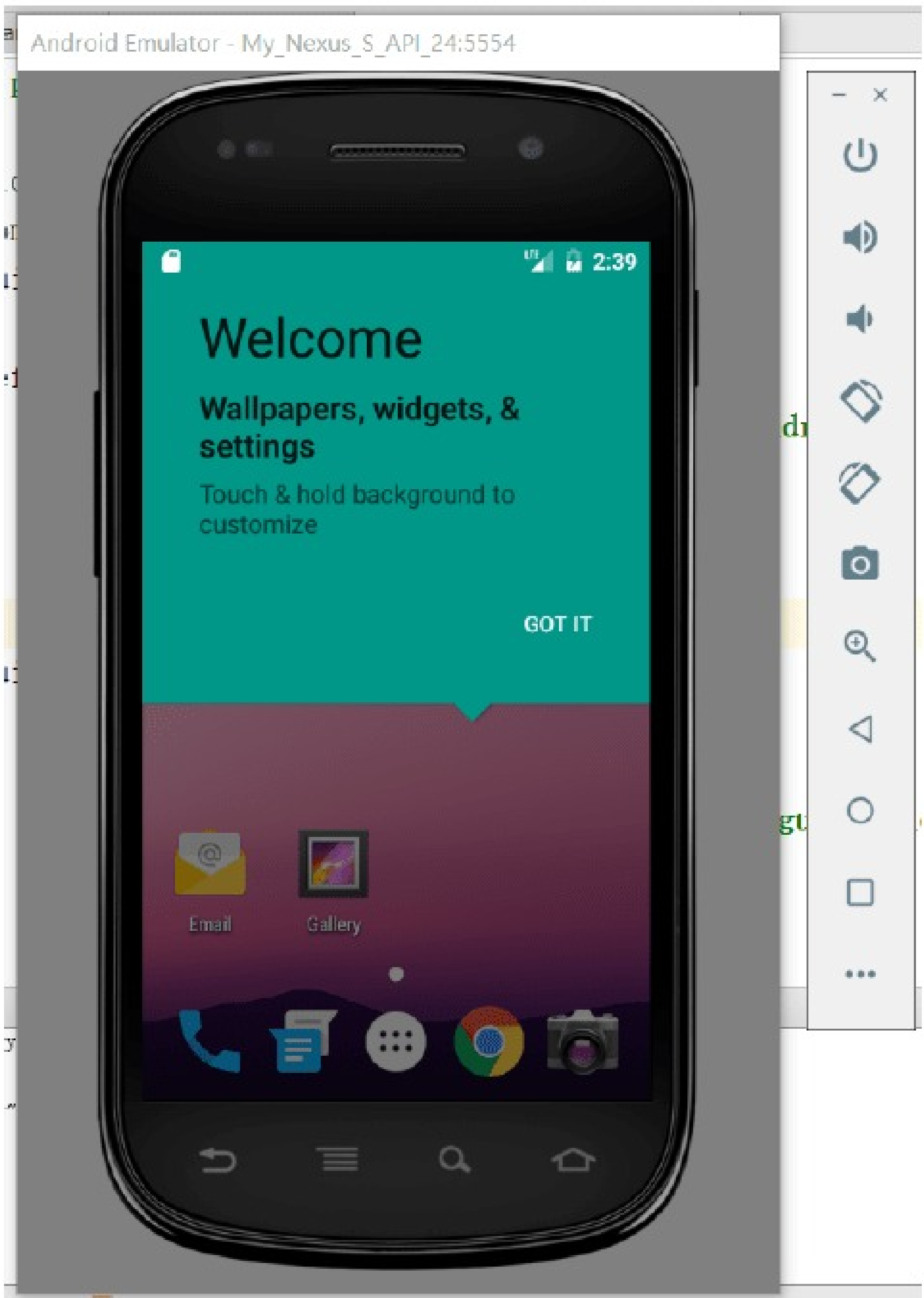


图 2.11　新创建的 AVD

2.2.4　AVD 与真机的区别

AVD 提供了近乎真实手机的虚拟环境，以便于程序员进行调试。但是 AVD 毕竟不是真机，有些功能目前 AVD 尚不能模拟，比如：

- AVD 不支持真实的电话接听和呼叫，但是可以通过控制台模拟电话呼叫。
- AVD 不支持 USB 连接。
- AVD 不支持相机/视频捕捉（输入）。
- AVD 不支持耳机。
- AVD 不支持蓝牙。
- AVD 不能在运行时确认 SD 卡的插入和弹出状态。
- AVD 不能确定电池的电量多少和充电状态。
- AVD 不能确定连接状态。

2.3 Android SDK 介绍

SDK（Software Development Kit）软件开发工具包是软件开发工程师用于为特定的软件包、软件框架、硬件平台、操作系统等建立应用软件的开发工具的集合。Android SDK 就是 Android 专属的软件开发工具包。

2.3.1 Android SDK 目录结构

Android SDK 解压即可完成安装，其中包含的文件、文件夹如图 2.12 所示。

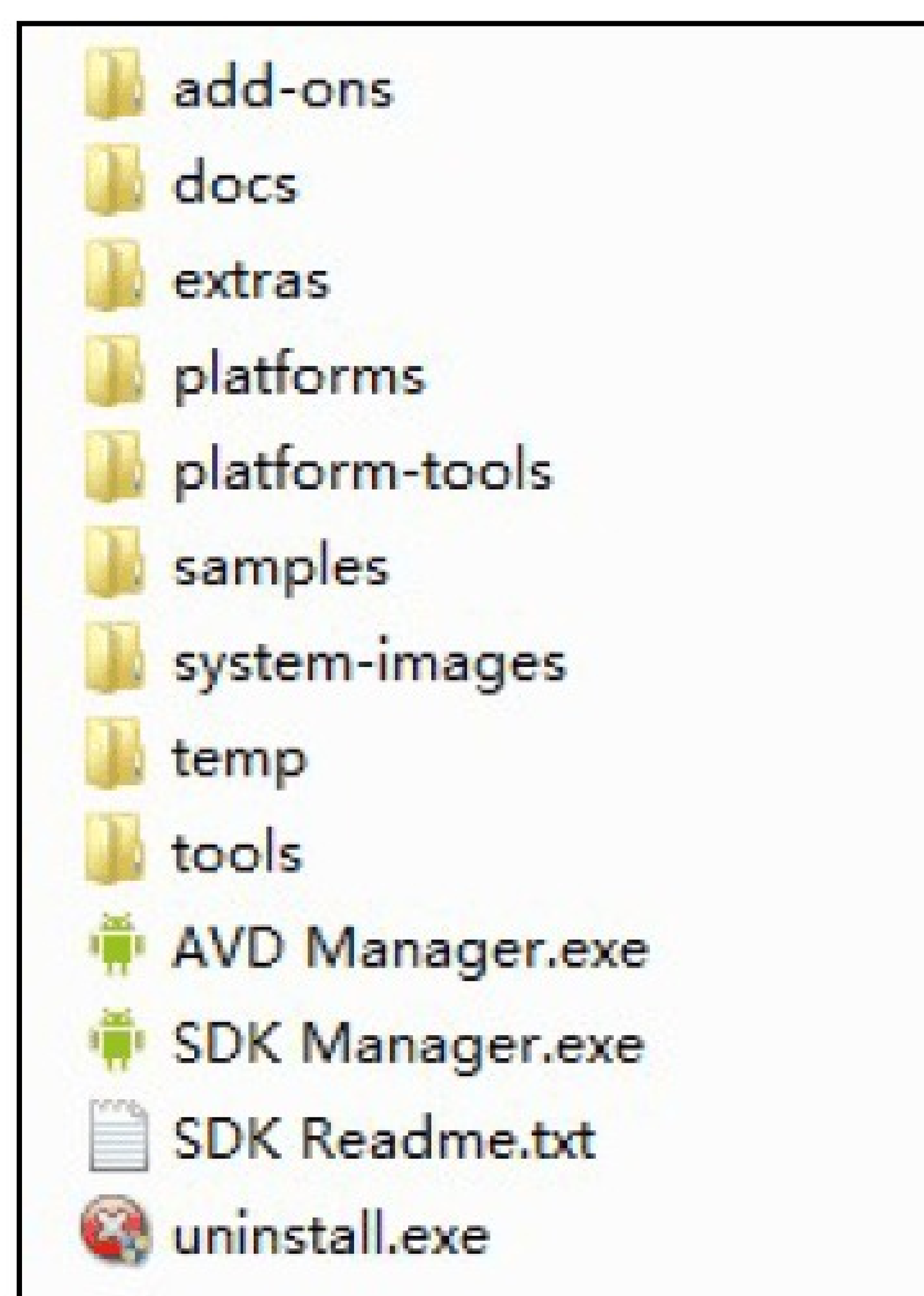


图 2.12 Android SDK 目录结构图

（1）add-ons

该目录中存放 Android 的扩展库，比如 Google Maps，但若未选择安装 Google API，则该目录为空。

（2）docs

该目录是 developer.Android.com 的开发文档，包含 SDK 平台、工具、ADT 等的介绍，开发指

南，API 文档，相关资源等。

（3）extras

该目录用于存放 Android 附加支持文件，主要包含 Android 的 support 支持包、Google 的几个工具和驱动、Intel 的 IntelHaxm。

（4）platforms

该目录用于存放 Android SDK Platforms 平台相关文件，包括字体、res 资源、模板等。

（5）platform-tools

该目录包含各个平台工具，其中主要包含以下几部分。

- api 目录。api-versions.xml 文件，用于指明所需类的属性、方法、接口等。
- lib 目录。lib 目录中只有 dx.jar 文件，为平台工具启动 dx.bat 时加载并使用 jar 包里的类。
- aapt.exe。主要作用是把开发的应用打包成 APK 安装文件，如果用 Eclipse 开发，就不用通过命令窗口输入命令+参数实现打包。
- adb.exe。ADB 即 Android Debug Bridge 调试桥，可以通过它连接 Android 手机（或模拟器）与 PC 端，可以在 PC 端上控制手机的操作。如果用 Eclipse 开发，一般情况下 ADB 会自动启动，之后我们可以通过 DDMS 来调试 Android 程序。
- aidl.exe。AIDL 全称是 Android Interface Definition Language，是 Android 内部进程通信接口的描述语言，用于生成可以在 Android 设备进行进程间通信（Inter-Process Communication，IPC）的代码。
- dexdump.exe。使用 dexdump 可以反编译.dex 文件，例如.dex 文件里包含 3 个类，反编译后也会出现 3 个.class 文件，通过这些文件可以大概了解原始的 Java 代码。
- dx.bat。其功能是将.class 字节码文件转成 Android 字节码.dex 文件。
- fastboot.exe。通过 Fastboot 可以进行重启系统、重写内核、查看连接设备、写分区、清空分区等操作。
- Android llvm-rs-cc.exe。Renderscript 采用 LLVM 低阶虚拟机，llvm-rs-cc.exe 的主要作用是对 Renderscript 的处理。
- NOTICE.txt 和 source.properties。NOTICE.txt 只是给出一些提示的信息；source.properties 是资源属性信息文件，主要显示该资源生成时间、系统类型、资源 URL 地址等。

（6）samples

samples 是 Android SDK 自带的默认示例工程，里面的 apidemos 强烈推荐初学者学习。

（7）system-images

该目录存放系统用到的所有图片。

（8）temp

该目录存放系统中的临时文件。

（9）tools

作为 SDK 根目录下的 tools 文件夹，这里包含重要的工具，比如 ddms 用于启动 Android 调试工具，如 logcat、屏幕截图和文件管理器；而 draw9patch 则是绘制 Android 平台的可缩放 PNG 图片的工具；sqlite3 可以在 PC 上操作 SQLite 数据库；而 monkeyrunner 则是一个不错的压力测试应用，模拟用户随机按钮；mksdcard 是模拟器 SD 映像的创建工具；emulator 是 Android 模拟器主程

序，不过从 Android 1.5 开始，需要输入合适的参数才能启动模拟器；traceview 是 Android 平台上重要的调试工具。

2.3.2 Android.jar

作为一个 Java 项目，通常情况下都会引入要用到的工具类，也就是 JAR 包，在 Android 开发中，绝大部分开发用的工具包都被封装到一个名叫 Android.jar 的文件里了。在 Eclipse 中展开来看，可以看到 J2SE 中的包、Apache 项目中的包，还有 Android 自身的包文件。Android 的包文件主要包括以下内容。

- Android.app：提供高层的程序模型和基本的运行环境。
- Android.content：包含各种对设备上的数据进行访问和发布的类。
- Android.database：通过内容提供者浏览和操作数据库。
- Android.graphics：底层的图形库。
- Android.location：定位和相关服务的类。
- Android.media：提供一些类管理多种音频、视频的媒体接口。
- Android.net：提供帮助网络访问的类，超过通常的 java.net.*接口。
- Android.os：提供系统服务、消息传输、IPC 机制。
- Android.openg ：提供 OpenGL 的工具。
- Android.provider：提供类，访问 Android 的内容提供者。
- Android.telephony：提供与拨打电话相关的 API 交互。
- Android.view：提供基础的用户界面接口框架。
- Android.util：涉及工具性的方法，例如时间日期的操作。
- Android.webkit：默认浏览器操作接口。
- Android.widget：包含各种 UI 元素（大部分是可见的）在应用程序的屏幕中使用。

2.3.3 Android API 核心包

SDK 中集成了很多开发应用的 API，它们通过 Android SDK 来编写应用程序的基础，这里从最底层到最高层列出核心包并加以说明。

- Android.util：包含一些底层辅助类，例如特定的容器类、XML 辅助工具类等。
- Android.os：提供基本的操作服务，如消息传递和进程间通信 IPC。
- Android.graphics：作为图形渲染包，提供图形渲染功能。
- Android.text Android.text.method Android.text.style Android.text.util：提供一套丰富的文本处理工具，支持富文本、输入模式等。
- Android.database：包含底层 API 处理数据库，方便操作数据库表和数据。
- Android.content：提供各种服务访问手机设备上的其他应用的数据和资料的接口。
- Android.view：核心用户界面框架。
- Android.widget：提供标准用户界面元素，如 List（列表）、Buttons（按钮）、Layout manager

（布局管理器）等，是组成界面的基本元素。

- Android.app：提供高层应用程序模型，实现使用 Activity。
- Android.provider：提供方便调用系统提供的 content providers 接口。
- Android.telephony：提供 API 和手机设备的通话接口。
- Android.webikit：包含一系列基于 Web 内容工作的 API。

2.3.4　Android API 扩展包

核心的 Android API 在每部手机上都可以使用，但仍然有一些 API 接口有各自特别的适用范围，这就是所谓的“可选 API”。这些 API 之所以是“可选的”，主要是因为一个手持设备并不一定要完全支持这类 API，甚至可以完全不支持。

- Location-Based Services（定位服务）。Android 操作系统支持 GPS API-LBS，可以通过集成 GPS 芯片来接收卫星信号，通过 GPS 全球定位系统中至少 3 颗卫星和原子钟来获取当前手机的坐标数据，通过转换就可以成为地图上的具体位置，这一误差在手机上可以缩小到 10 米。在谷歌开发手机联盟中可以看到著名的 SiRF star。所以未来 gPhone 手机上市时集成 GPS 后的价格不会很贵。同时，谷歌正在研制基于基站式的定位技术-MyLocation，可以更快速地定位，与前者 GPS 定位需要花费大约 1 分钟相比，基站定位更快。
- Media APIs（多媒体接口）。Android 平台上集成了很多影音解码器以及相关的多媒体 API，通过这些可选 API，厂商可以让手机支持 MP3、MP4、高清晰视频播放处理等。
- 3D Graphics with OpenGL（3D 图形处理 OpenGL），可选 API。Android 平台上的游戏娱乐功能，如支持 3D 游戏或应用场景就需要用到 3D 技术，手机生产厂商根据手机的屏幕以及定位集成不同等级的 3D 加速图形芯片来加强 gPhone 手机的娱乐性，有来自高通的消息称，最新的显示芯片在 gPhone 上将会轻松超过索尼 PS3。
- Low-Level Hardware Access（低级硬件访问）。这个功能主要用于控制手机的底层方面操作，设计底层硬件操作将主要由各个手机硬件生产厂商来定制，支持不同设备的操作管理，如蓝牙（Bluetooth）以及 WIFI 无线网络支持等。

2.4　创建第一个 Android 应用程序

2.4.1　创建 HelloWorld 工程

启动 Android Studio，依次选择 File | New | New Project，将会出现如图 2.13 所示的界面。在 Application name 中输入项目名称“HelloWorld”，在 Company Domain 中输入“android.introduction”，系统会自动生成包名为“introduction.Android.helloWorld”，Project Location 指定工程文件存放的位置。单击“Next”按钮，出现如图 2.14 所示的界面，用于选择应用运行的系统版本。选择运行平台为“Android 7.0”，再次单击“Next”按钮，进入创建 Activity 界面，如图 2.15 所示。该界面

可以添加多种 Activity 的模板，本次添加一个基本的 Activity 即可，选择“Basic Activity”，单击“Next”按钮，进入如图 2.16 所示的界面，指定 Activity 的相关信息，例如 Activity 的名字、布局文件的名字、菜单资源的名字以及 Activity 上显示的标题。此处使用默认设置，不做更改。设置完成后，单击“Finish”按钮完成工程的创建。

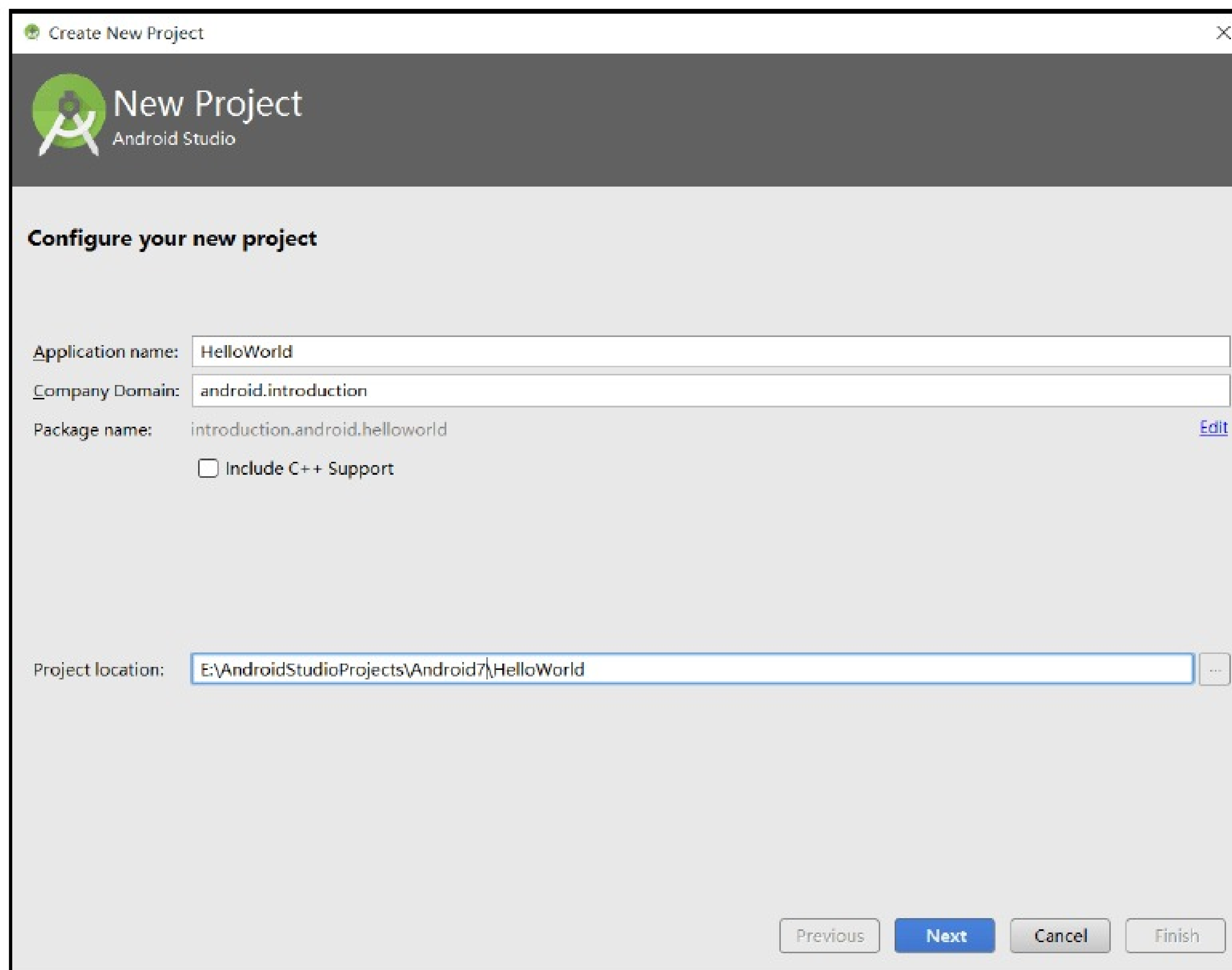


图 2.13　创建 HelloWorld 工程

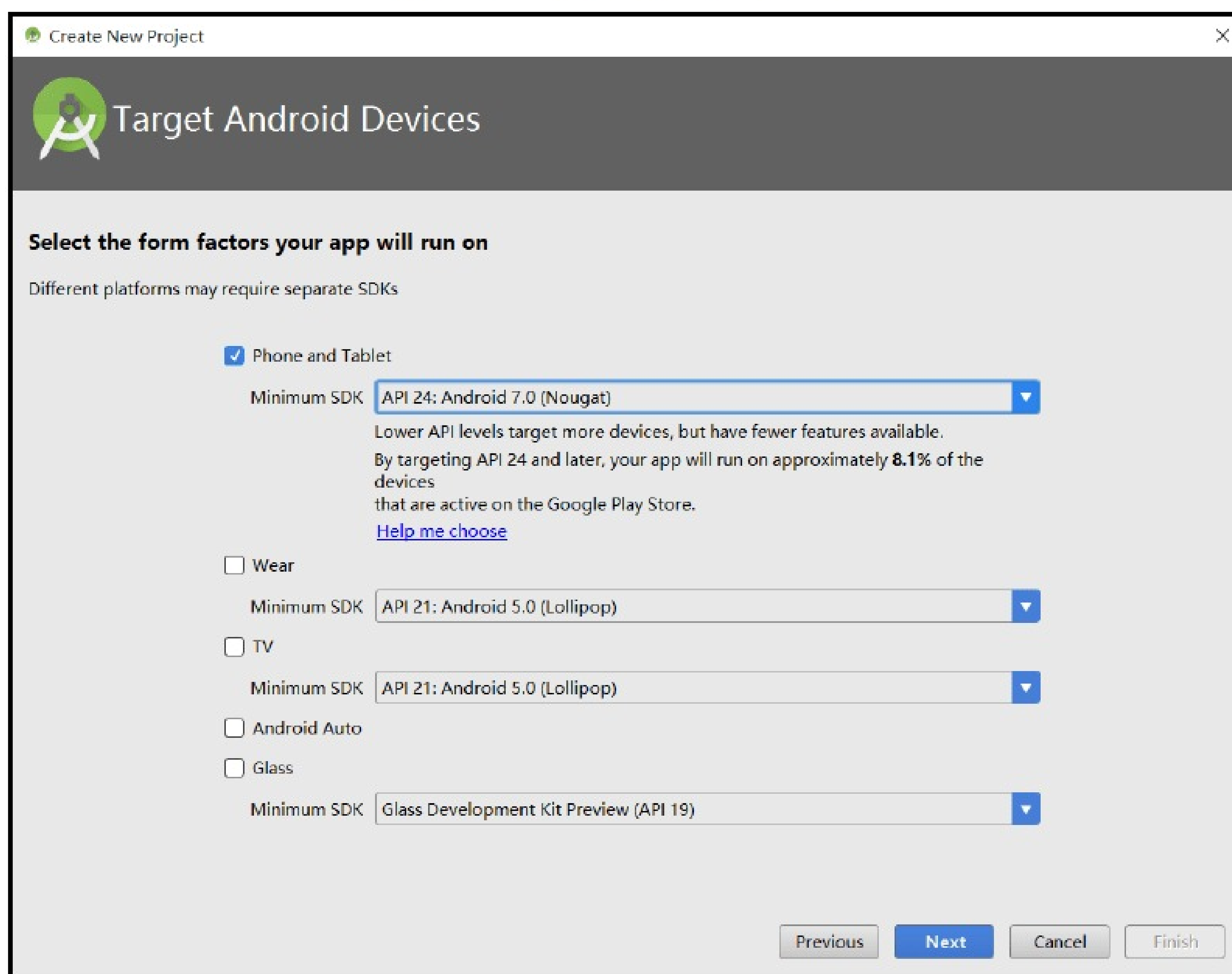


图 2.14　选择应用系统平台

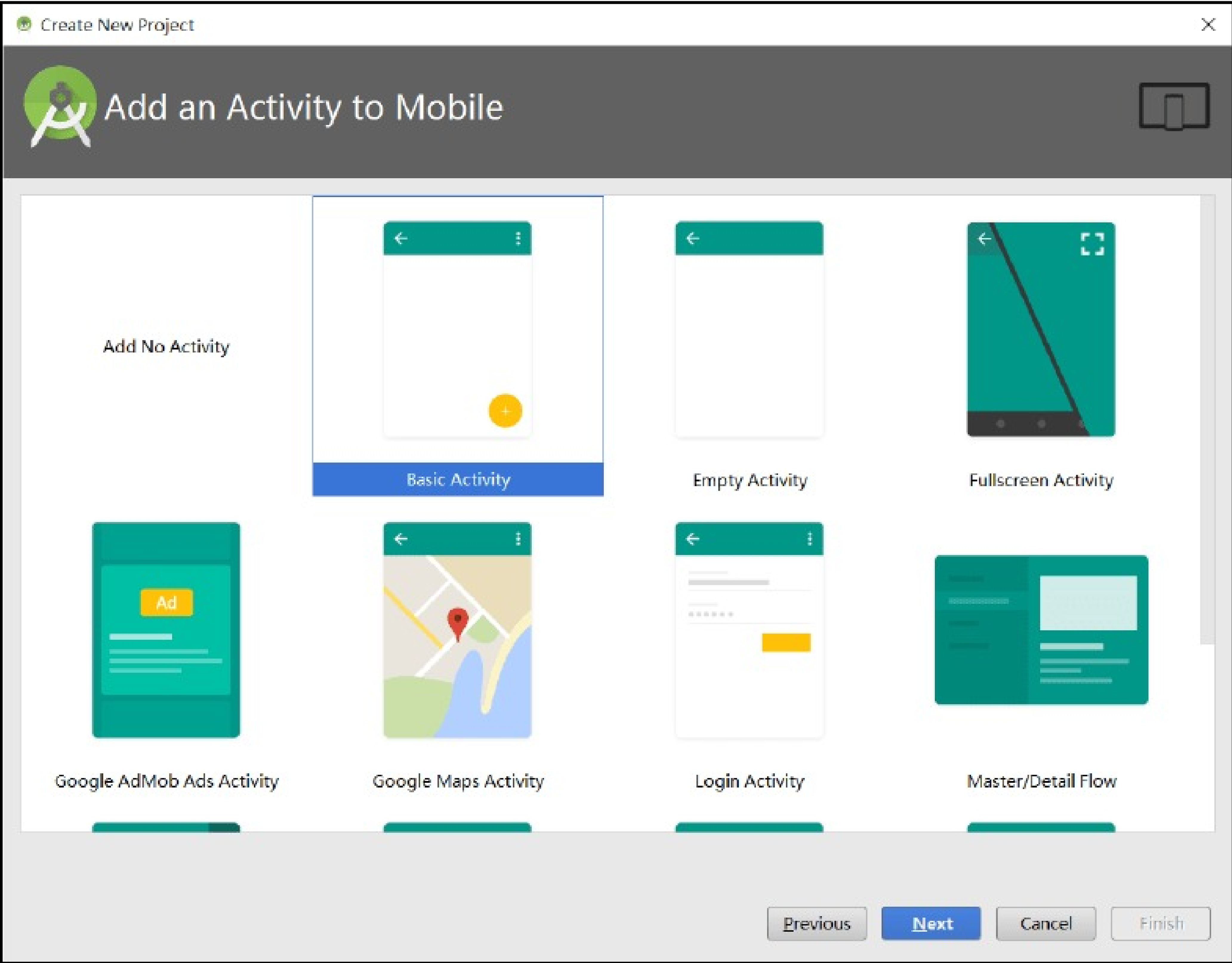


图 2.15　创建 Activity

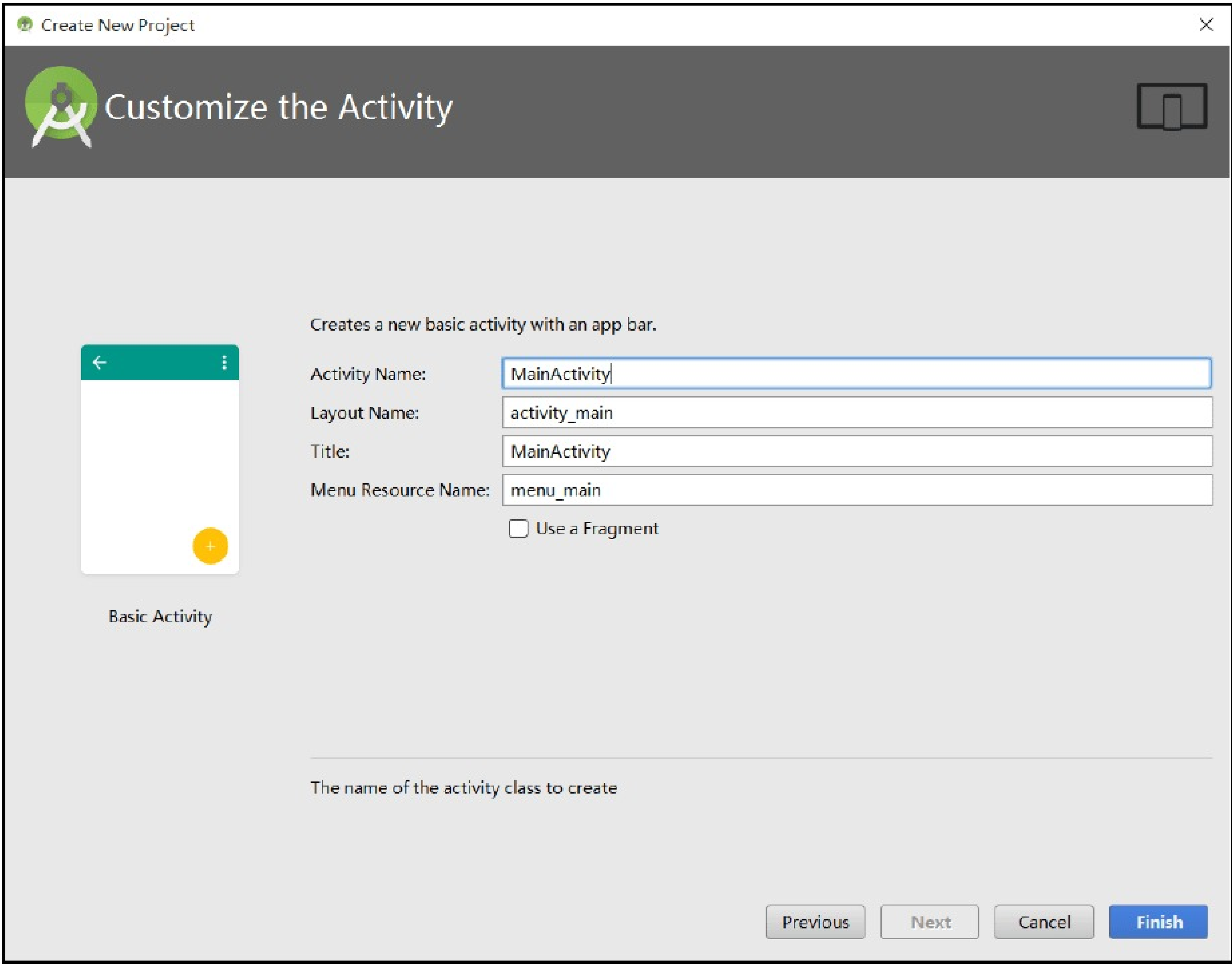


图 2.16　指定 Activity 的相关信息

Android Studio 会根据刚才指定的相关信息生成相关模板代码，用户无须编写任何一行代码，该工程就可以运行。按 Shift+F10 快捷键，选择要运行的 AVD，可查看运行效果，如图 2.17 所示。

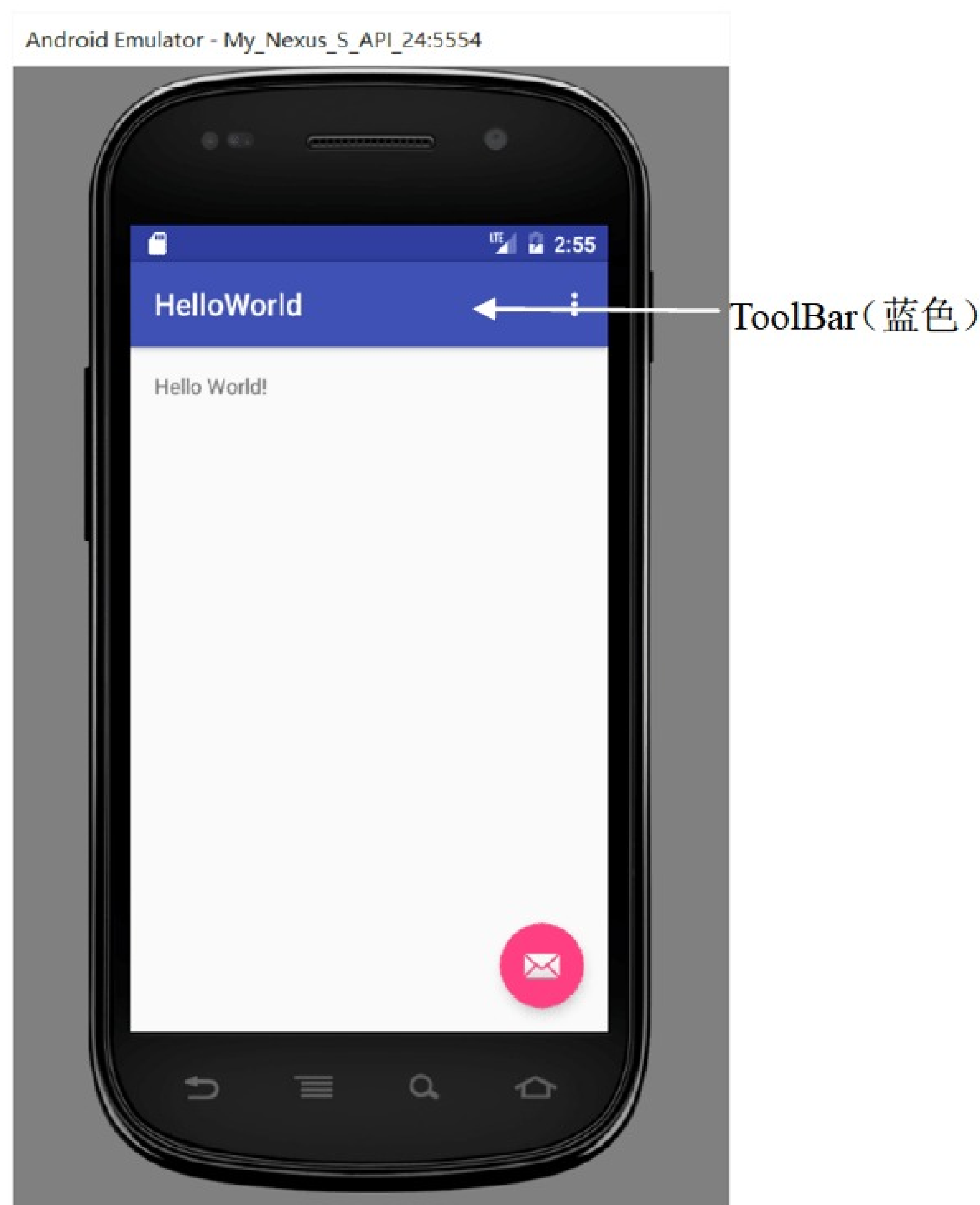


图 2.17　运行效果（效果图颜色可在下载资源中查看）

2.4.2　相关代码

双击 HelloWorld 工程中的 MainActivity.java，该文件中已有程序代码如下：

```
package introduction.android.helloworld;

import android.os.Bundle;
import android.support.design.widget.FloatingActionButton;
import android.support.design.widget.Snackbar;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.Toolbar;
import android.view.View;
import android.view.Menu;
import android.view.MenuItem;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);

        FloatingActionButton fab = (FloatingActionButton) findViewById(R.id.fab);
        fab.setOnClickListener(new View.OnClickListener() {
```

```
            @Override
            public void onClick(View view) {
                Snackbar.make(view, "Replace with your own action", Snackbar.LENGTH_LONG)
                        .setAction("Action", null).show();
            }
        });
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.menu_main, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        // Handle action bar item clicks here. The action bar will
        // automatically handle clicks on the Home/Up button, so long
        // as you specify a parent activity in AndroidManifest.xml.
        int id = item.getItemId();

        //noinspection SimplifiableIfStatement
        if (id == R.id.action_settings) {
            return true;
        }

        return super.onOptionsItemSelected(item);
    }
}
```

MainActivity.java 中的代码比较简单，表明类 MainActivity 继承了 AppCompatActivity 类，并重写了 onCreate()方法。

AppCompatActivity 类是 Android Studio 中默认的构建自定义 Activity 的模板类，与 Eclipse+ADT 环境中默认使用的 Activity 相比，AppCompatActivity 提供了对工具栏 ToolBar 的支持，相关代码如下：

```
Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
```

在 MainActivity 的 onCreate()方法体中调用了父类的 onCreate()方法，然后调用 setContentView()方法显示视图界面。Android 工程中使用 XML 文件来设计视图界面，R.layout.activity_main 是 Android 工程中默认的布局文件的名字，即 activity_main.xml。

Activity_main.xml 的内容如下：

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.design.widget.CoordinatorLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:fitsSystemWindows="true"
```

```
        tools:context="introduction.android.helloworld.MainActivity">

        <android.support.design.widget.FloatingActionButton
            android:id="@+id/fab"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="bottom|end"
            android:layout_margin="@dimen/fab_margin"
            app:srcCompat="@android:drawable/ic_dialog_email" />

        <android.support.design.widget.AppBarLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:theme="@style/AppTheme.AppBarOverlay">

            <android.support.v7.widget.Toolbar
                android:id="@+id/toolbar"
                android:layout_width="match_parent"
                android:layout_height="?attr/actionBarSize"
                android:background="?attr/colorPrimary"
                app:popupTheme="@style/AppTheme.PopupOverlay" />

        </android.support.design.widget.AppBarLayout>

        <include layout="@layout/content_main" />

    </android.support.design.widget.CoordinatorLayout>
```

CoordinatorLayout 布局是 support v7 系统新增的布局，具有便于调度协调子布局的特点。该布局可看作是增强版的 FrameLayout，通常与 ToolBar 和 FloatingActionButton 合用。

ToolBar 是图 2.17 中显示 HelloWorld 的蓝色工具栏，具有承载系统菜单的功能。布局相关代码如下：

```
        <android.support.design.widget.AppBarLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:theme="@style/AppTheme.AppBarOverlay">

            <android.support.v7.widget.Toolbar
                android:id="@+id/toolbar"
                android:layout_width="match_parent"
                android:layout_height="?attr/actionBarSize"
                android:background="?attr/colorPrimary"
                app:popupTheme="@style/AppTheme.PopupOverlay" />
        </android.support.design.widget.AppBarLayout>
```

FloatingActionButton 是图 2.17 中右下侧的邮箱图标的按钮，布局相关代码如下：

```
    <android.support.design.widget.FloatingActionButton
            android:id="@+id/fab"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="bottom|end"
            android:layout_margin="@dimen/fab_margin"
            app:srcCompat="@android:drawable/ic_dialog_email" />
```

在 MainActivity.java 中，FloatingActionButton 的事件处理代码为：

```
FloatingActionButton fab = (FloatingActionButton) findViewById(R.id.fab);
       fab.setOnClickListener(new View.OnClickListener() {
          @Override
          public void onClick(View view) {
             Snackbar.make(view, "Replace with your own action", Snackbar.LENGTH_LONG)
                    .setAction("Action", null).show();
          }
});
```

该代码实现的功能是，当点击按钮时，显示“Replace with your own action”。

```
<include layout="@layout/content_main" />
```

这行代码将 content_main.xml 的布局嵌入 activity_main 布局中。content_main.xml 的代码为：

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:app="http://schemas.android.com/apk/res-auto"
   xmlns:tools="http://schemas.android.com/tools"
   android:id="@+id/content_main"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:paddingBottom="@dimen/activity_vertical_margin"
   android:paddingLeft="@dimen/activity_horizontal_margin"
   android:paddingRight="@dimen/activity_horizontal_margin"
   android:paddingTop="@dimen/activity_vertical_margin"
   app:layout_behavior="@string/appbar_scrolling_view_behavior"
   tools:context="introduction.android.helloworld.MainActivity"
   tools:showIn="@layout/activity_main">

   <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Hello World!" />
</RelativeLayout>
```

该文件中的代码表示当前的布局文件使用 LinearLayout 布局，该布局中仅有一个 TextView 组件用于显示信息，显示的内容为“Hello World！”。

Android Studio 鼓励用户将所有组件放置到 content_main.xml 中，而对 activity_main 中的代码尽量不做修改。

为了简化代码，降低阅读难度，在本书的范例程序代码中，除非需要用到工具栏和悬浮按钮，都会将.java 文件和.xml 文件中的 ToolBar 和 FloatingActionButton 的相关代码移除掉，并且直接使用单个布局文件搭建界面，避免使用 include 将一个布局嵌入另一个布局中。

2.4.3　工程文件结构解析

没有书写一句程序代码，一个 Android 应用便创建成功了，但是这只是一个简单的 Android 应用，要创建更多的 Android 应用，还要详细地了解 Android 应用程序结构。

Android Studio 的 Project 工程文件结构如图 2.18 所示。

主要目录的作用如下。

- .gradle 目录：Gradle 在构建工程的过程中生成的文件。
- .idea 目录：Android Studio 生成的工程配置文件，类似 Eclipse 的 project.properties。
- build 目录：相当于 Eclipse 工程的 bin 目录，用于存放生成的文件，包括 APK。
- gradle 目录：用于存放 Gradle 构建工具系统的 JAR 和 Wrapper 等，以及配置文件。
- External Libraries：工程依赖的 LIB 文件，如 SDK 等。
- app 目录：Android Studio 创建工程中的一个 Module，是程序开发者的主要工作目录。app 目录下的结构如图 2.19 所示。

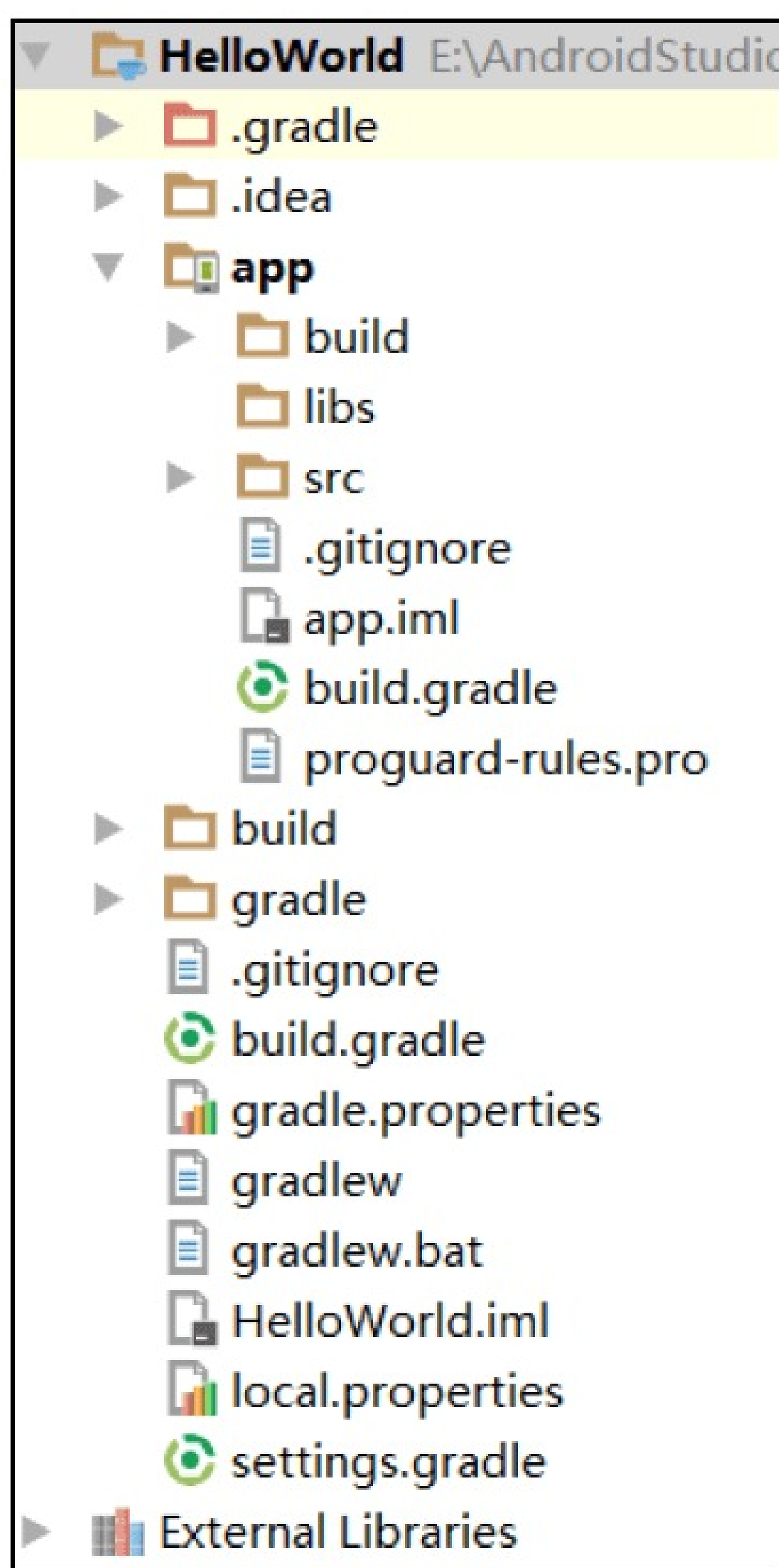


图 2.18　Android Studio 工程文件结构

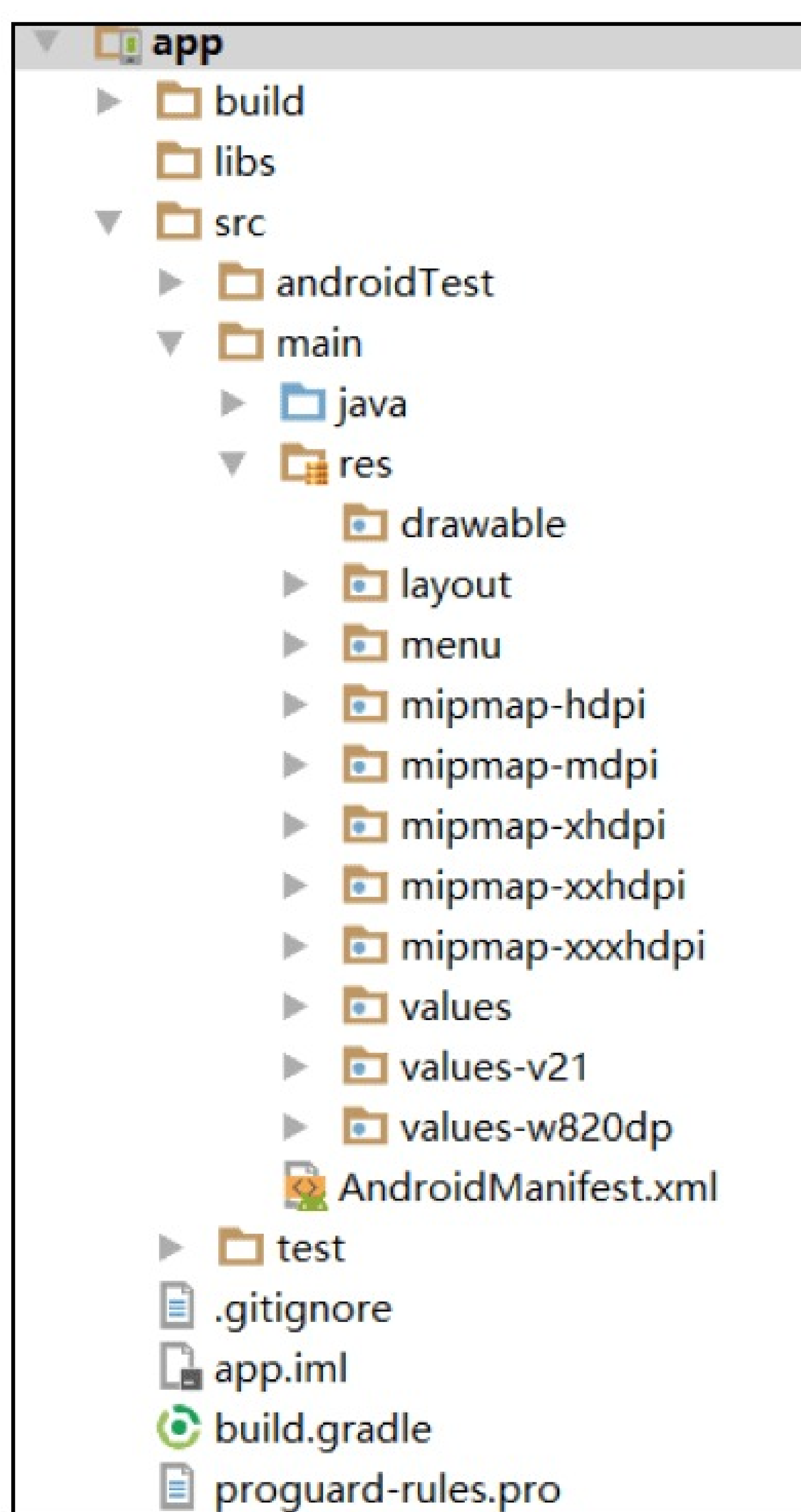


图 2.19　app 目录结构

下面分别介绍各个目录或文件的作用。

- src。该目录（文件夹）中包含应用程序的所有源代码。在 src 文件夹中可以创建若干 Java 包，在包中可以创建应用的处理逻辑以及应用的 Activity，MainActivity.java 就是在创建项目的时候创建的一个 Activity，在 Activity 中可以编写控制 View 的逻辑。
- build。该目录（文件夹）的 source 包中有一个“R.java”文件。R 类中包含 4 个静态内部类：attr、drawable、layout 和 string，分别代表属性、图片资源、布局文件及字符串的声明。R.java 文件是资源索引类，由 Eclipse 自动生成，开发者不用去修改和维护里面的内容，但是这个文件却非常有用，它和 res 文件夹紧密相连，对 res 下资源的操作都会导致 R.java 文件的重新编译。R.java 中定义的常量类也可以间接帮助 Activity 完成对资源的应用。Android 这样设计的好处是使得复杂的资源通过专门的类来管理而让程序中的代码变得整齐、强壮，并且减少程

序出错和 bug 的产生。

- assets。该目录（文件夹）中通常放置一些原始资源文件，它会在 Android 打包的时候原封不动地一起打包，安装时会直接解压到对应的 assets 目录中。这里通常放置一些项目中用到的多媒体资源等。
- res。目录（文件夹）中放置的是 Android 要用到的各种程序资源。其中，常见的子文件夹有 drawable、layout、values 等。其中，drawable 目录放置应用到的图片资源；layout 目录放置一些与 UI 相关的布局文件，都是以 XML 文件方式保存; values 目录中放置的是一些字符串、数组、颜色、样式和动画等资源，values 目录中的每一个文件都会转化成 R.java 中的一个静态类，文件中的每一个资源都会转化成 R.java 中对应静态类的静态整型常量，这样 Activity 中通过一个解析器就可以获取对应的资源。
- AndroidManifest.xml。这个文件是整个项目的配置资源，里面配置的内容包括当前应用程序所在的包、应用程序中的 Activity、应用程序的访问权限等。

2.5　调试程序

2.5.1　设置断点

设置断点检查每个变量的运行输出更适合一些大型项目的排错或状态检测，是 Java 开发中不可缺少的调试方法。

设置断点的方法有两种：

（1）双击 Android Studio 代码编辑区左边的区域。

（2）在需要添加或者移除断点的代码处按 Ctrl+F8 快捷键。

2.5.2　调试

通过单击工具栏上的按钮，或者在项目上右击，然后选择 Debug …菜单命令，或者按 Shift+F9 快捷键，启动程序的调试模式，如图 2.20 所示。

当程序运行到设置的断点时就会停下，这时可以按照下面的功能键按需求进行调试：

- 快捷键 F8 单步执行程序。
- 快捷键 F7 单步执行程序，遇到方法时进入。
- 快捷键 Alt+F9 运行到光标处。

在调试界面，变量的值会出现在 Variables 窗口中，这样就可以查看运行至断点时变量当前的值，可在 Watches 界面添加想观测的变量或者对象的值。

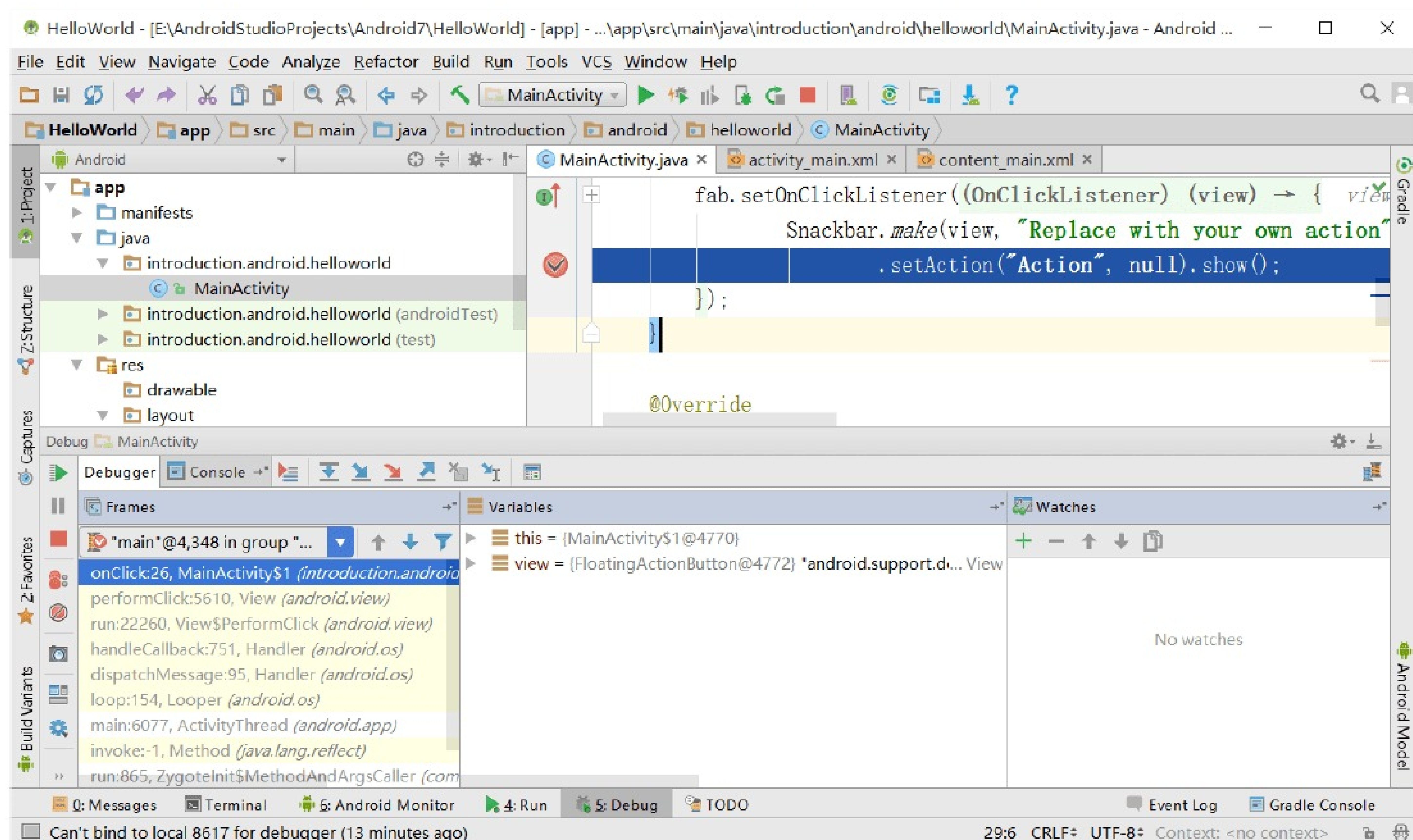


图 2.20 调试界面

2.6 小 结

本章主要介绍了 Android 开发环境的搭建，并以 HelloWorld 为例讲解了 Android 工程的创建过程。Android 工程文件结构主要包括 src、gen、res、Android 目录以及 AndroidManifest.xml 文件，开发者应该熟知每个工程目录的作用。还介绍了 Android Studio 开发平台的基本调试方法，希望读者在日后的学习中能够熟悉应用程序的调试方法。此外，本章还介绍了 Android SDK 的目录结构及其核心包和扩展包。

2.7 习 题

1. 尝试创建自己的第一个 Android 工程。
2. 请简要介绍 Android 工程中各目录的作用。
3. 谈谈你对 Android SDK 的认识。

第 3 章

Android 应用程序结构

Android 作为一个移动设备的开发平台，其软件层次结构包含操作系统（OS）、中间件（MiddleWare）和应用程序（Application）。其中，Android 的应用程序通常涉及用户界面和用户交互，这类程序是用户实实在在能感受到的，目前 Android 本身提供了桌面、联系人、电话和浏览器等众多的核心应用，同时还允许开发者使用应用程序框架层的 API 实现自己的程序。

3.1 应用程序基本组成

Android 系统没有使用常见的应用程序入口点的方法（例如 main()方法），应用程序是由组件组成的，组件可以调用相互独立的基本功能模块，根据完成的功能不同，Android 划分了 4 类核心组件，即 Activity、Service、BroadcastReceiver 和 ContentProvider，各组件之间的消息传递通过 Intent 完成。

3.1.1 Activity

Activity 是 Android 应用程序核心组件中最基本的一种，是用户和应用程序交互的窗口。在 Android 应用程序中，一个 Activity 通常对应一个单独的视图。一个 Android 应用程序是由一个或多个 Activity 组成的，这些 Activity 相当于 Web 应用程序中的网页，用于显示信息，并且相互之间可以进行跳转。和网页跳转不同的是，Activity 之间的跳转可以有返回值。

当新打开一个视图时，之前的那个视图会被置为暂停状态，并且压入历史堆栈中，用户可以通过回退操作返回以前打开过的视图。Activity 是由 Android 系统进行维护的，它有自己的生命周期，即“产生、运行、销毁”，但是在这个过程中会调用许多方法，如创建 onCreate()、激活 onStart()、恢复 onResume()、暂停 onPause()、停止 onStop()、销毁 onDestroy()和重启 onRestart()等。

3.1.2 Service

Service 是一种类似于 Activity 但是没有视图的程序，它没有用户界面，可以在后台运行很长时间，相当于操作系统中的一个服务。Android 定义了两种类型的 Service，即本地 Service 和远程 Service。本地 Service 是只能由承载该 Service 的应用程序访问的组件，而远程 Service 是供在设备上运行的其他应用程序远程访问的 Service。

通过 Context.startService(Intent service)可以启动一个 Service，通过 Context. bindService()可以绑定一个 Service。

3.1.3 BroadcastReceiver

BroadcastReceiver 的意思是“广播接收者”，顾名思义，它用来接收来自系统和其他应用程序的广播，并做出回应。在 Android 系统中，当有特定事件发生时就会产生相应的广播。广播体现在方方面面，例如，当开机过程完成后，系统会产生一条广播，接收到这条广播就能实现开机启动服务的功能；当网络状态改变时，系统会产生一条广播，接收到这条广播就能及时地做出提示和保存数据等操作；当电池电量改变时，系统会产生一条广播，接收到这条广播就能在电量低时告知用户及时保存进度等。

BroadcastReceiver 不能生成 UI，通过 NotificationManager 来通知用户有事件发生，对于用户来说是隐式的。BroadcastReceiver 的注册方式有两种，一种是在 AndroidManifest. xml 中进行静态注册；另一种是在运行时的代码中使用 Context.registerReceiver()进行动态注册。只要注册了 BroadcastReceiver，即使对应的事件广播来临时应用程序并未启动，系统也会自动启动该应用程序对事件进行处理。另外，用户还可以通过 Context.sendBroadcast()将自己的 Intent 对象广播给其他的应用程序。

3.1.4 ContentProvider

文件、数据库等数据在 Android 系统内是私有的，仅允许被特定应用程序直接使用。在两个程序之间，数据的交换或共享由 ContentProvider 实现。

ContentProvider 类实现了一组标准方法的接口，从而能够让其他的应用保存或读取 ContentProvider 提供的各种数据类型。

3.1.5 Intent

Intent 并不是 Android 应用程序四大核心组件之一，但是其重要性无可替代，因此在这里我们做一下简单介绍。

Android 应用程序核心组件中的三大核心组件——Activity、Service、BroadcastReceiver，通过消息机制被启动激活，而所使用的消息就是 Intent。Intent 是对即将要进行的操作的抽象描述，承担了 Android 应用程序三大核心组件相互之间的通信功能。

3.2 Activity

Activity 是 Android 组件中最基本也是最为常见的组件。Activity 是用户接口程序，原则上它会提供给用户一个交互式的接口功能，几乎所有的 Activity 都要和用户打交道，也有人把它比喻成 Android 的管理员。需要在屏幕上显示什么、用户在屏幕上做什么、处理用户的不同操作等都由 Activity 来管理和调度。

Activity 提供用户与 Android 系统交互的接口，用户通过 Activity 来完成自己的目的，例如打电话、拍照、发送 E-mail、查看地图等。每个 Activity 都提供一个用户界面窗口，一般情况下，该界面窗口会填满整个屏幕，但是也可以比屏幕小，或者浮在其他的窗口之上。

一个 Android 应用程序通常由多个 Activity 组成，但是其中只有一个为主 Activity，其作用相当于 Java 应用程序中的 main 函数，当应用程序启动时，作为应用程序的入口首先呈现给用户。Android 应用程序中的多个 Activity 可以直接相互调用以完成不同工作。当新的 Activity 被启动的时候，之前的 Activity 会停止，但是不会被销毁，而是被压入“后退栈（Back Stack）”的栈顶，新启动的 Activity 获得焦点，显示给用户。“后退栈”遵循“后入先出”的原则。当新启动的 Activity 被使用完毕，用户单击“Back”按钮时，当前的 Activity 会被销毁，而原先的 Activity 会被从“后退栈”的栈顶弹出并且激活。

当 Activity 状态发生改变时，都会通过状态回调函数通知 Android 系统。而程序编写人员可以通过这些回调函数对 Activity 进行进一步的控制。

下面对 Activity 生命周期及其涉及的回调函数进行简单介绍。

3.2.1 Activity 的生命周期

从本质上讲，Activity 在生命周期中共存在三个状态，分别如下。

- 运行态：运行态指 Activity 运行于屏幕的最上层并且获得了用户焦点。
- 暂停态：暂停态是指当前 Activity 依然存在，但是没有获得用户焦点。在其之上有其他的 Activity 处于运行态，但是由于处于运行态的 Activity 没有遮挡住整个屏幕，当前 Activity 有一部分视图可以被用户看见。处于暂停态的 Activity 保留了自己所使用的内存和用户信息，但是在系统极度缺乏资源的情况下，有可能会被杀死以释放资源。
- 停止态：停止态是指当前 Activity 完全被处于运行态的 Activity 遮挡住，其用户界面完全不能被用户看见。处于停止态的 Activity 依然存活，也保留了自己所使用的内存和用户信息，但是一旦系统缺乏资源，停止态的 Activity 就会被杀死以释放资源。

Activity 在生命周期中从一种状态到另一种状态时会激发相应的回调方法，这些回调方法包括：

- onCreate（Bundle savedInstanceState）。创建 Activity 时调用。设置在该方法中，还以 Bundle 的形式提供对以前储存的任何状态的访问。其中参数 savedInstanceState 对象是用于保存 Activity 的对象的状态。
- onStart()。Activity 变为在屏幕上对用户可见时调用。
- onResume()。Activity 开始与用户交互时调用（无论是启动还是重启一个活动，该方法总是被调用）。

- onPause()。当 Android 系统要激活其他 Activity 时，该方法被调用，暂停或收回 CPU 和其他资源时调用。
- onStop()。Activity 被停止并转为不可见阶段时调用。
- onRestart()。重新启动已经停止的 Activity 时调用。
- onDestroy()。Activity 被完全从系统内存中移除时调用，该方法被调用可能是因为有人直接调用 finish()方法或者系统决定停止该活动以释放资源。

上面 7 个生命周期方法分别在 4 个阶段按着一定的顺序进行调用，这 4 个阶段如下：

- 启动 Activity。在这个阶段依次执行 3 个生命周期方法：onCreate、onStart 和 onResume。
- Activity 失去焦点。如果在 Activity 获得焦点的情况下进入其他的 Activity 或应用程序，这时当前的 Activity 会失去焦点。在这一阶段，会依次执行 onPause 和 onStop 方法。
- Activity 重获焦点。如果 Activity 重新获得焦点，会依次执行 3 个生命周期方法：onRestart、onStart 和 onResume。
- 关闭 Activity。当 Activity 被关闭时，系统会依次执行 3 个生命周期方法：onPause、onStop 和 onDestroy。

Activity 生命周期中方法的调用过程如图 3.1 所示。

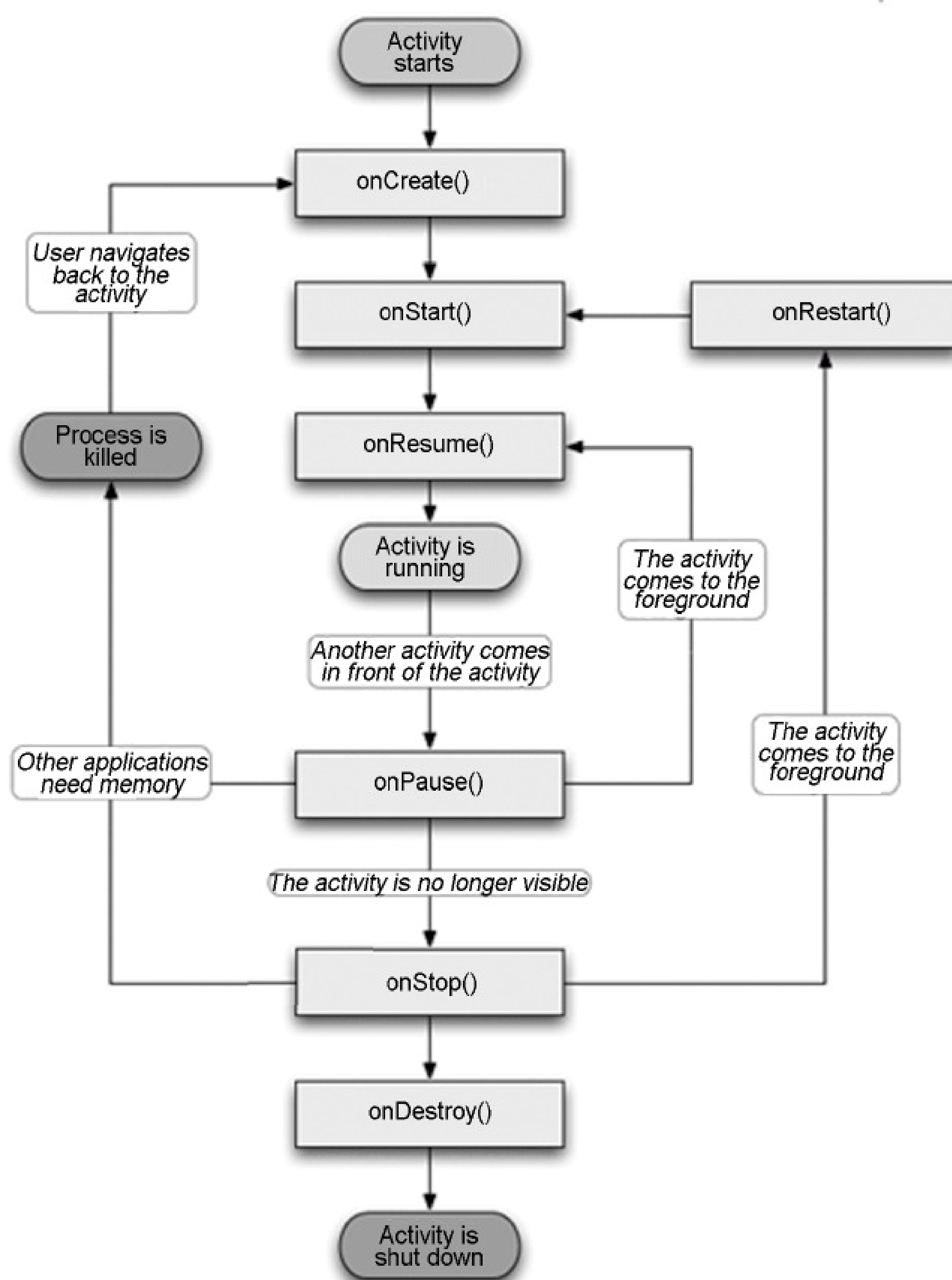


图 3.1　Activity 生命周期

通过图 3.1，可以很直观地了解到 Activity 的整个生命周期。Activity 的生命周期表现在三个层面，如图 3.2 所示。

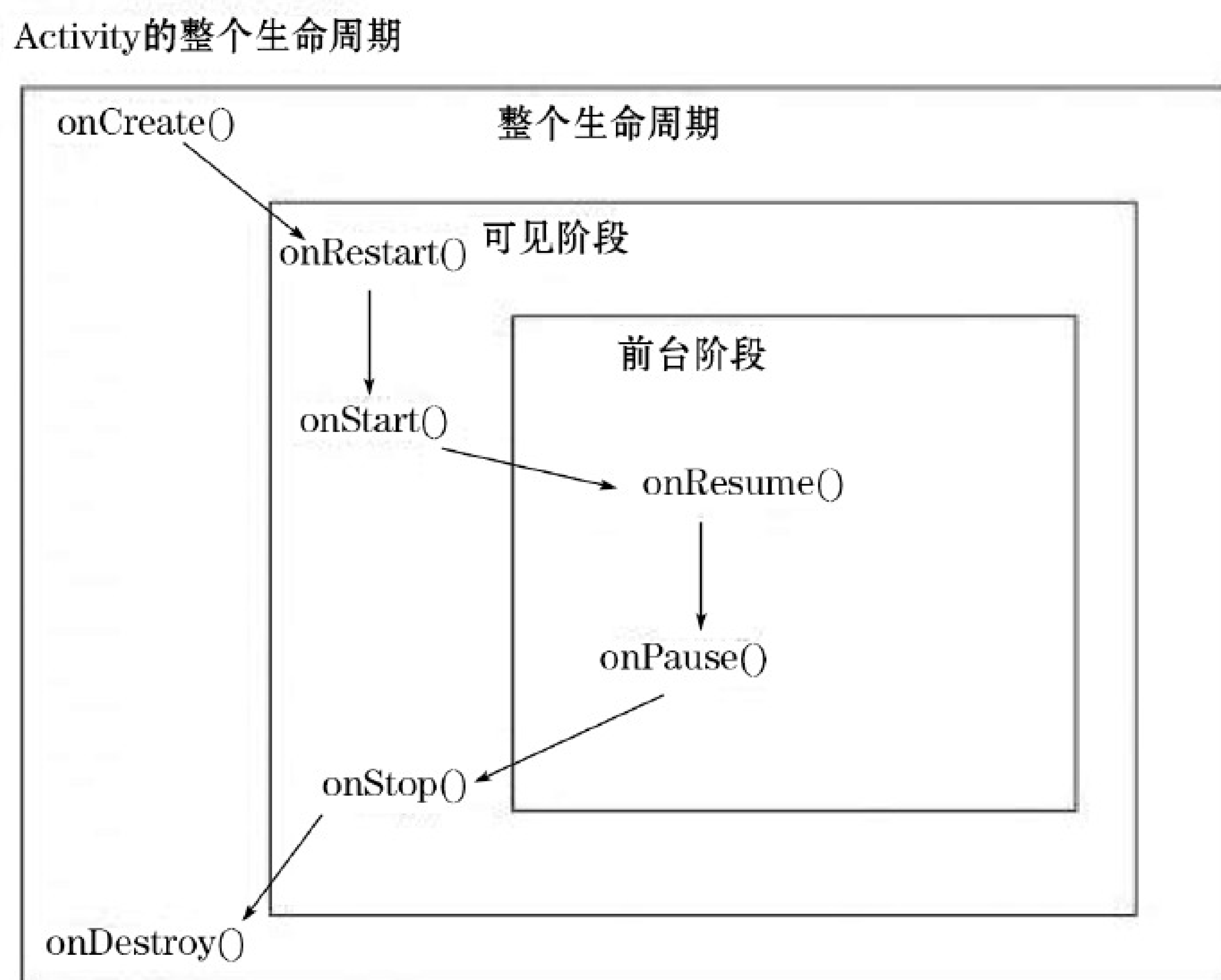


图 3.2　Activity 的整个生命周期

通过图 3.2 可以更清楚地了解 Activity 的运行机制。如果 Activity 离开可见阶段，长时间失去焦点，就很可能被系统销毁以释放资源。当然，即使该 Activity 被销毁掉，用户对该 Activity 所做的更改也会被保存在 Bundle 对象中，当用户需要重新显示该 Activity 时，Android 系统会根据之前保存的用户更改信息将该 Activity 重建。

3.2.2　Activity 的创建

在一个 Android 工程中，创建 Activity 的步骤如下：

步骤 01　新建类。创建一个 Activity，必须创建 Android.app.Activity（或者它的一个已经存在的子类）的一个子类，并重写 onCreate()方法。

步骤 02　关联布局 XML 文件。在新建的 Activity 中设置其布局方式，需要在 res/layout 目录中新建一个 XML 布局文件，可以通过 setContentView()来指定 Activity 的用户界面的布局文件。

步骤 03　注册。在 AndroidManifest.xml 文件中对建立的 Activity 进行注册，即在<application>标签下添加<activity>标签。例如，注册 ExampleActivity 的代码如下：

```
<application ...>
     <activity Android:name=".ExampleActivity" />
      ...
</application ...>
```

对于主 Activity，要为其添加<intent-filter>标签，代码如下：

```
<activity Android:name=".ExampleActivity" Android:icon="@drawable/app_icon">
   <intent-filter>
       <action Android:name="Android.intent.action.MAIN" />
```

```
        <category Android:name="Android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

其中，<action Android:name="Android.intent.action.MAIN" />表示该 Activity 作为主 Activity 出现，而<category Android:name="Android.intent.category.LAUNCHER" />表示该 Activity 会被显示在最上层的启动列表中。

3.2.3 启动 Activity

在 Android 系统中，除了主 Activity 由系统启动外，其他 Activity 都要由应用程序来启动。

（1）通常情况下，通过 startActivity()方法来启动 Activity，而要启动的 Activity 的信息由 Intent 对象来传递，例如：

```
Intent intent=new Intent(this, AnotherActivity.class);
startActivity(intent);
```

表示通过当前的 Activity 启动名为 AnotherActivity 的 Activity。

有时，用户不需要知道要启动的 Activity 的名字，而可以仅制定要完成的行为，由 Android 系统来为用户挑选合适的 Activity，例如：

```
Intent intent=new Intent(Intent.ACTION_SEND);
intent.putExtra(Intent.EXTRA_EMAIL, recipientArray);
startActivity(intent);
```

其中，Intent.EXTRA_EMAIL 放置的是 recipientArray 中存储的要发送的 E-mail 的目标地址。该 Intent 对象被 startActivity()启动后，Android 系统会启动相应的 E-mail 处理应用程序，并将 Intent.EXTRA_EMAIL 中的内容放置到邮件的目标地址中。

（2）有时，当需要从启动的 Activity 获取返回值的时候，需要使用 startActivityForResult()方法代替 startActivity()方法，并实现 onActivityResult()方法来获取返回值。

例如，在发送短信的时候，用户需要从联系人列表中获取联系人的信息，然后返回到短信发送界面，代码如下：

```
Intent intent=new Intent(Intent.ACTION_PICK, Contacts.CONTENT_URI);
startActivityForResult(intent, PICK_CONTACT_REQUEST);
```

当用户选择了联系人后，相关信息会被存储到 Intent 对象中，并返回到 onActivityResult()方法中。

3.2.4 关闭 Activity

关闭 Activity 使用 finish()方法。关闭之前启动的其他 Activity 可以使用 finishActivity()方法。

需要注意的是，虽然 Android SDK 提供了关闭 Activity 的方法，但是通常情况下，程序员不应该使用这些方法去强制关闭 Activity。因为 Android 系统在为用户维护 Activity 的生命周期，并且提供了完备的资源回收机制和资源重建机制，可以动态地回收和重建 Activity，因此 Activity 应用交由 Android 系统来管理，除非已确定用户不再需要当前的 Activity，并且不允许用户回退到当前 Activity。

3.2.5　Activity 数据传递

Activity 数据传递共有三种：

- 通过 Intent 传递一些简单的数据。
- 通过 Bundle 传递相对复杂的数据或者对象。
- 通过 startActivityForResult 可以更方便地进行来回传递，当然前两种方法也可以来回传递。

假设由 Activity1 向 Activity2 传递数据，利用三种方式实现的实例代码如下。

（1）利用 Intent 传递数据。

在传递数据的 Activity1 中：

```
Intent intent=new Intent(Activity1.this,Activity2.class);
intent.putExtra("author","leebo");//在 Intent 中加入键值对数据，键为“author”，值为“leebo”
Activity1.this.startActivity(intent);
```

在取出数据的 Activity2 中：

```
Intent intent=getIntent();//获得传过来的 Intent
String value=intent.getStringExtra("author");
//根据键名 author 取出对应键值为“leebo”
```

（2）利用 Bundle 传递数据。

在传递数据的 Activity1 中：

```
Intent intent=new Intent(Activity1.this,Activity2.class);
Bundle myBundle=new Bundle();
myBundle.putString("author","leebo");
intent.putExtras(myBundle);
Activity1.this.startActivity(intent);
```

在取出数据的 Activity2 中：

```
Intent intent=getIntent();
Bundle myBundle=intent.getExtras();
String value=myBundle.getString("author"); //根据键名 author 取出对应键值为“leebo”
```

（3）利用 startActivityForResult()传递数据。

startActivityForResult()方法不但可以把数据从 Activity1 传递给 Activity2，还可以把数据从 Activity2 传回给 Activity1。

在 Activity1 中：

```
final int REQUEST_CODE=1;
Intent intent=new Intent(Activity1.this,Activity2.class);
Bundle mybundle=new Bundle();
mybundle.putString("author", "leebo");//把数据传过去
intent.putExtras(mybundle);
startActivityForResult(intent, REQUEST_CODE);
```

重载 onActivityResult 方法，用来接收传过来的数据（接收 b 中传过来的数据）：

```
 protected void onActivityResult(int requestCode, int resultCode,Intent intent){
     if(requestCode==this.REQUEST_CODE){
```

```
        switch (resultCode) {
        case RESULT_OK:
           Bundle b=intent.getExtras();
           String str=b.getString("Result"); //获取 Result 中的值，为“from Activity2”
            break;
       default:
             break;
       }
       }
     }
```

在 Activity2 中：

```
     Intent intent=getIntent();
     Bundle myBundle=getIntent().getExtras();
     String author=getBundle.getString("author");
     Intent intent=new Intent();
     Bundle bundle=new Bundle();
     bundle.putString("Result","from Activity2");
     intent.putExtras(bundle);
     Activity02.this.setResult(RESULT_OK,intent);//通过 intent 将数据返回给 Activity1，RESULT_ OK
是结果码:
     finish();//结束当前的 Activity
```

本质上，这三种数据传递方式都是通过 Intent 来完成的。

3.3 资　　源

在 Android 层次结构中，资源扮演着重要的角色。Android 支持字符串、位图以及其他很多种类型的资源。每一种资源的语法、格式以及存放的位置都会根据其类型的不同而不同。一般来讲，共有三种类型的资源文件：XML 文件、位图文件（图像）和 RAW 文件（声音等）。

Android 工程目录中，用于存放资源文件的文件夹有两个，分别为 res 和 assets。其中，res 文件夹不支持深度子目录，其中的资源最终将被打包到编译后的 Java 文件中，可以直接通过 R 资源类访问，利用率较高；而 assets 中存放的资源是用于打包到应用程序中的静态文件，这些文件不会被编译，最终会直接部署到目标设备中，可以使用任意深度的子目录进行存储。assets 文件夹中的文件不能直接通过 R 资源类读取，只能使用流的形式读取，其利用率相对较低。

Android 的资源编译器 AAPT（Android Asset Packaging Tool）会依照资源所在的子目录及其格式对其进行编译。

3.4 Manifest 文件

每一个 Android 项目都包含一个清单（Manifest）文件 AndroidManifest.xml，它是 XML 格式的 Android 程序声明文件，包含 Android 系统运行程序前所必须掌握的重要信息，这些信息包含应

用程序名称、图标、包名称、模块组成、授权和 SDK 最低版本等，而且每个 Android 程序必须在根目录下包含一个 AndroidManifest.xml。

例如，Manifest 文件可以使用如下代码声明一个 Activity：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ... >
   <application android:icon="@drawable/app_icon.png" ... >
      <activity android:name="com.example.project.ExampleActivity"
              android:label="@string/example_label" ... >
      </activity>
      ...
   </application>
</manifest>
```

AndroidManifest.xml 中可包含的所有标签元素如以下代码所示，其中除了<manifest>和<application>标签是必需的，其他所有标签都可按情况添加。

```
<?xml version="1.0" encoding="utf-8"?>
<manifest>
   <uses-permission />
   <permission />
   <permission-tree />
   <permission-group />
   <instrumentation />
   <uses-sdk />
   <uses-configuration />
   <uses-feature />
   <supports-screens />
   <compatible-screens />
   <supports-gl-texture />
   <application>
      <activity>
         <intent-filter>
            <action />
            <category />
            <data />
         </intent-filter>
         <meta-data />
      </activity>
      <activity-alias>
         <intent-filter>. . .</intent-filter>
         <meta-data />
      </activity-alias>
      <service>
         <intent-filter>. . .</intent-filter>
         <meta-data/>
      </service>
      <receiver>
         <intent-filter>. . .</intent-filter>
         <meta-data />
      </receiver>
      <provider>
         <grant-uri-permission />
         <meta-data />
```

```
        </provider>
        <uses-library />
    </application>
</manifest>
```

在此，仅对几种常见的标签进行简单介绍。

（1）manifest 标签

manifest 标签是 AndroidManifest.xml 文件的根标签，该标签用于设置与项目相关的一些属性，比如用于唯一标识应用程序的 package 属性，用于记录应用程序版本的 Android:versionName 属性，等等。其中的 xmlns:Android 属性必须被定义为“http://schemas. Android.com/apk/res/Android”。

（2）application 标签

manifest 标签仅能包含一个 application 标签，它使用各种属性来指定应用程序的各种元数据（包括标题、图标和主题）。它还可以作为一个包含活动（Activity）、服务（Service）、内容提供器（Provider）和广播接收器（Broadcast Receiver）标签的容器，用来指定应用程序组件。

- activity 标签。应用程序显示的每一个 Activity 都要求有一个 activity 标签，并使用 Android:name 属性来指定类的名称。这必须包含核心的启动 Activity 和其他所有可以显示的屏幕或者对话框。启动任何一个没有在清单中定义的 Activity 时都会抛出一个运行时异常。每一个 Activity 节点都允许使用 intent-filter 子标签来指定哪个 Intent 启动该活动。
- service 标签。和 activity 标签一样，应用程序中使用的每一个 Service 类都要创建一个新的 service 标签。（Service 标签也支持使用 intent-filter 子标签来允许后面的运行时绑定。）
- provider 标签。provider 标签用来说明应用程序中的每一个内容提供器。内容提供器是用来管理数据库访问以及程序内和程序间共享的。
- receiver 标签。通过添加 receiver 标签，可以注册一个广播接收器，而不用事先启动应用程序。广播接收器就像全局事件监听器一样，一旦注册了之后，无论何时，只要与它相匹配的 Intent 被应用程序广播出来，它就会立即执行。通过在声明中注册一个广播接收器，可以使这个进程实现完全自动化。如果一个匹配的 Intent 被广播了，应用程序就会自动启动，并且你注册的广播接收器也会开始运行。

（3）uses-permission 标签

作为安全模型的一部分，uses-permission 标签声明了那些自己定义的权限，而这些权限是应用程序正常执行所必需的。在安装程序时，设定的所有权限将会告诉用户，由他们来决定同意与否。对很多本地 Android 服务来说，权限都是必需的，特别是那些需要付费或者有安全问题的服务（例如拨号、接收 SMS 或者使用基于位置的服务）。第三方应用程序，包括你自己的应用程序，也可以在提供对共享的程序组件进行访问之前指定权限。

（4）permission 标签

在可以限制访问某个应用程序组件之前，需要在清单中定义一个 permission。可以使用 permission 标签来创建这些权限定义。然后，应用程序组件就可以通过添加 Android: permission 属性来要求这些权限。其他的应用程序需要在它们的清单中包含 uses-permission 标签（并且通过授权），之后才能使用这些受保护的组件。在 permission 标签内，可以详细指定允许的访问权限的级别（normal、dangerous、signature 和 signatureOrSystem）、一个 label 属性和一个外部资源，这个外部资源应该包含对授予这种权限的风险的描述。

（5）instrumentation 标签

instrumentation 类提供一个框架，用来在应用程序运行时在活动或者服务上运行测试。它们提供了一些方法来监控应用程序及其与系统资源的交互。对于为自己的应用程序所创建的每一个测试类，都需要创建一个新的节点。

3.5　App Widgets

App Widgets 是指能够嵌入其他应用程序中的小组件，并且能够周期性地进行更新。App Widgets 并不是 Android 应用程序的核心组件，但却是应用程序开发不可或缺的部分。我们可以通过 App Widgets 使我们的 UI 界面更多样化，也可以通过 App Widget Provider 发布我们自己开发的 App Widgets 组件。一个能够用于容纳 App Widgets 组件的应用程序组件被称为 App Widgets Host（App Widgets 宿主），例如图 3.3 所示的音乐播放程序。



图 3.3　App Widgets Host

Android 7.0 中涉及部分 App Widgets 类的使用方法会在第 4 章进行详细介绍，本节主要对使用 App Widget Provider 发布自己的 App Widget 组件的方法进行简单介绍。

3.5.1　基础知识

为了创建一个自己的 App Widget，需要完成以下工作。

1. AppWidgetProviderInfo 元数据

定义在 XML 文件中的用于描述 App Widget 的元数据对象，比如 App Widget 的布局、更新频率以及相关的 AppWidgetProvider 类。

2. 实现 AppWidgetProvider 类

在 AppWidgetProvider 类中定义了一系列方法，这些方法允许开发者以编程的方式和自己的 App Widget 进行交互，这种交互基于广播事件。当 App Widget 的状态发生改变，例如更新、启用、禁用和删除的时候，你都会接收到相应的广播通知。

3. 视图布局

在 XML 文件中为 App Widget 定义初始布局。

4. 实现 App Widget 配置 Activity

这是一个可选的 Activity，当用户添加 App Widget 时该 Activity 会被启动，并允许用户在创建

App Widget 时修改相关设置。

下面进行详细介绍。

3.5.2 在 Manifest 文件中声明 App Widget

首先，在 AndroidManifest.xml 文件中对 AppWidgetProvider 类进行声明。相关代码如下：

```
    <receiver android:name="ExampleAppWidgetProvider" >
   <intent-filter>
       <action android:name="android.appwidget.action.APPWIDGET_UPDATE" />
   </intent-filter>
   <meta-data android:name="android.appwidget.provider"
             android:resource="@xml/example_appwidget_info" />
</receiver>
```

<receiver>元素必须要指定 android:name 属性，它指定了 App Widget 使用的 AppWidgetProvider 的名字。

<intent-filter>元素必须包括一个含有 android:name 属性的<action>元素。该元素指定 AppWidgetProvider 接受 ACTION_APPWIDGET_UPDATE 广播。这是唯一一个必须被显式声明的广播。当有必要的时候，AppWidgetManager 会自动发送所有其他 App Widget 广播给 AppWidgetProvider。

<meta-data>元素指定了 AppWidgetProviderInfo 资源并需要以下属性。

- android:name：指定元数据名称。
- android:resource：指定 AppWidgetProviderInfo 资源路径。

3.5.3 增加 AppWidgetProviderInfo 元数据

AppWidgetProviderInfo 用于定义 App Widget 的一系列基本特性，例如最小布局的尺寸、初始的布局资源、刷新频率以及创建时要加载的配置 Activity 等。使用<appwidget-provider>元素标签在 XML 中定义 AppWidgetProviderInfo 对象并保存到项目的 res/xml/目录下，例如：

```
    <appwidget-provider xmlns:android="http://schemas.android.com/apk/res/android"
        android:minWidth="294dp" <!-- density-independent pixels -->
        android:minHeight="72dp"
        android:updatePeriodMillis="86400000" <!-- once per day -->
        android:initialLayout="@layout/example_appwidget"
        android:configure="com.example.android.ExampleAppWidgetConfigure" >
    </appwidget-provider>
```

其中：

- minWidth 和 minHeight 属性的值指定了这个 App Widget 布局需要的最小区域。
- updatePerdiodMillis 属性定义了 App Widget 框架调用 onUpdate()方法来从 AppWidgetProvider 请求一次更新的频率。实际上更新的时间并不精准。建议更新频率越低越好，比如一小时更新一次，这样可以节省电力，或者根据用户的配置调整更新频率，比如有个人每 15 分钟想查

看一下股票的报价，这样可以将频率设置为一小时更新 4 次。

- initialLayout 属性指向 App Widget 使用的布局的资源。
- configure 属性定义了该 App Widget 被加载时使用的配置 Activity。

3.5.4　创建 App Widget 布局

必须在 res/layout 目录下以 XML 文件的方式为 App Widget 定义一个布局文件。App Widget 的布局是基于 RemoteViews 对象的，而 RemoteViews 对象可以支持以下布局：

- Framelayout
- LinearLayout
- RelativeLayout
- GridLayout

和以下的小组件类：

- AnalogClock
- Button
- Chronometer
- ImageButton
- ImageView
- ProgressBar
- TextView
- ViewFlipper
- ListView
- GridView
- StackView
- AdapterViewFlipper

但是并不支持它们的派生类。

此外，RemoteView 还支持 ViewStub，该组件不可见，自身无尺寸，可用于对布局资源进行支撑。

3.5.5　为 App Widget 添加边界

如果没有为自定义的 Widget 定义边界，它就会自动扩展到屏幕大小。因此，我们需要为自定义的 App Widget 定义边界。

自 Android 4.0 开始，App Widget 会自动在 Widget 的边界环绕盒之间添加空隙，以便为 Widget 和其他小组件以及屏幕上的图标提供更好的排列组合方式。为实现这个行为，我们需要将应用程序中的“ targetSdkVersion”属性设置为大于 14。

实际上，我们可以自己定义一个带有自定义边界的布局，并且使该布局在应用于早期平台版

本时正常显示边界，而在 Android 4.0 以后版本的平台上不显示额外边界。定义过程如下：

步骤 01 设置 targetSdkVersion 为大于 14 的值。

步骤 02 创建一个布局，并为其设置 dimension 资源，其边界信息由 dimension 资源设定，代码如下：

```
  <FrameLayout
 android:layout_width="match_parent"
 android:layout_height="match_parent"
 android:padding="@dimen/widget_margin">

 <LinearLayout
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="horizontal"
   android:background="@drawable/my_widget_background">
   …
 </LinearLayout>

</FrameLayout>
```

步骤 03 创建两个 dimension 资源，一个在 res/values/目录下，用于提供低于 Android 4.0 版本的系统的边界信息，另一个在 res/values-v14 下，用于提供高于 Android 4.0 版本的操作系统的边界信息。

例如，res/values/dimens.xml 定义如下：

```
<dimen name="widget_margin">8dp</dimen>
```

而 res/values-v14/dimens.xml 定义如下：

```
<dimen name="widget_margin">0dp</dimen>
```

3.5.6 使用 AppWidgetProvider 类

首先，AppWidgetProvider 类是 BroadcastReceiver 类的子类，可以方便地处理 App Widget 发出的广播，因此，其必须被声明在清单文件中的<receiver>元素中。AppWidgetProvider 只接受和相应的 App Widget 相关的广播消息，例如这个 App Widget 被更新、被删除、被启用或者被禁用的时候。当这些广播事件发生的时候，AppWidgetProvider 会接收到以下方法的调用请求。

- onUpdate()：每间隔一定时间该方法就会被调用用于对 App Widget 进行更新。间隔时间由 AppWidgetProviderInfo 元数据中的 updatePeriodMillis 属性指定。当用户添加 App Widget 时，该方法也会被调用。因此，该方法中应该执行必要的操作，例如为视图定义事件处理器或者启动一个临时的服务等。如果你为 App Widget 定义了配置 Activity，就应该由配置 Activity 负责进行第一次更新，而 onUpdate()方法不会在用户执行添加操作的时候被调用，而只会在后期的更新时被调用。

- onAppWidgetOptionsChanged()：该方法在 Widget 被首次放置到应用程序中或者 Widget 的尺寸被更改时被调用。
- onDeleted(Context, int[])：该方法在 App Widget 被从 App Widget 宿主中删除的时候被调用。
- onEnabled(Context)：该方法在 App Widget 的第一个实例被创建时被调用。若用户添加了两个 App Widget 的实例，则该方法只会在第一次添加时被调用。如果你需要打开数据库或者其他只需要进行一次的设置，那么将代码放在这个方法中是个不错的主意。
- onDisabled(Context)：该方法在最后一个 App Widget 实例从 App Widget 宿主中被删除的时候调用。在该方法中，你应该对在 onEnabled()方法中的操作进行善后，例如删除一个临时的数据库。
- onReceive(Context, Intent)：每当接收到一个广播，该方法都会被调用。并且，该方法会在上述各个方法之前被调用。通常我们不需要重写该方法，因为默认的 AppWidgetProvider 类已经很好地实现了对所有广播的过滤和处理方法的调用。

可见 onUpdate()方法是最重要的回调方法，如果你创建的 App Widget 不需要进行创建临时文件等操作，那么你可能只需要定义 onUpdate()方法就可以了。例如，当你创建了一个带有 Button 的 App Widget，当点击按钮时会启动一个 Activity，那么你的 AppWidgetProvider 类应该像下面这样定义：

```
    public class ExampleAppWidgetProvider extends AppWidgetProvider {

   public void onUpdate(Context context, AppWidgetManager appWidgetManager, int[] appWidgetIds)
{
        final int N = appWidgetIds.length;

        // Perform this loop procedure for each App Widget that belongs to this provider
        for (int i=0; i<N; i++) {
            int appWidgetId = appWidgetIds[i];

            // Create an Intent to launch ExampleActivity
            Intent intent = new Intent(context, ExampleActivity.class);
            PendingIntent pendingIntent = PendingIntent.getActivity(context, 0, intent, 0);

            // Get the layout for the App Widget and attach an on-click listener
            // to the button
            RemoteViews views = new RemoteViews(context.getPackageName(),
R.layout.appwidget_provider_layout);
            views.setOnClickPendingIntent(R.id.button, pendingIntent);

            // Tell the AppWidgetManager to perform an update on the current app widget
            appWidgetManager.updateAppWidget(appWidgetId, views);
        }
    }
}
```

其中，appWidgetIds 是一个存放 ID 的数组，其中的每一个 ID 值都标识一个 AppWidgetProvider 创建的 App Widget。如果该数组中存放了多个 App Widget 的 ID，那么这些 App Widget 会被同步更新。

3.5.7 接收 App Widget 的广播

如果你想直接用自己的类接收并处理 App Widget 的广播，那么你需要实现自己的 BroadcastReceiver，重写 onReceiver()方法，并处理以下 4 个 Intent：

- ACTION_APPWIDGET_UPDATE
- ACTION_APPWIDGET_DELETED
- ACTION_APPWIDGET_ENABLED
- ACTION_APPWIDGET_DISABLED

3.5.8 创建 App Widget 的配置 Activity

如果想让用户在添加新的 App Widget 的时候对颜色、尺寸、更新周期等属性进行配置，那么就需要创建一个配置 Activity。配置 Activity 会在 App Widget 被创建时由其宿主启动。

该配置 Activity 需要在 Manifest 文件中进行声明，通过 ACTION_APPWIDGET_CONFIGURE 活动被宿主启动，代码如下：

```
<activity android:name=".ExampleAppWidgetConfigure">
    <intent-filter>
        <action android:name="android.appwidget.action.APPWIDGET_CONFIGURE" />
    </intent-filter>
</activity>
```

此外，该 Activity 还需要在 AppWidgetProviderInfo XML 中通过 android:configure 属性被声明，例如：

```
<appwidget-provider xmlns:android="http://schemas.android.com/apk/res/android"
    ...
    android:configure="com.example.android.ExampleAppWidgetConfigure"
    ... >
</appwidget-provider>
```

当为 App Widget 定义了配置 Activity 后，Widget 在被创建时不会再调用 onUpdate 方法。

3.5.9 使用配置 Activity 对 App Widget 进行更新

当 Widget 使用了配置 Activity 后，配置 Activity 会在用户完成设置后对 Widget 进行更新。通过配置 Activity 对 Widget 进行更新并关闭配置 Activity 的过程如下：

步骤 01 **从启动 Activity 的 Intent 中获取 App Widget 的 ID 值。**

```
    Intent intent = getIntent();
Bundle extras = intent.getExtras();
if (extras != null) {
   mAppWidgetId = extras.getInt(
           AppWidgetManager.EXTRA_APPWIDGET_ID,
           AppWidgetManager.INVALID_APPWIDGET_ID);
}
```

步骤02 执行 App Widget 配置。

步骤03 完成配置后，获取 AppWidgetManager 类的实例。

```
    AppWidgetManager appWidgetManager = AppWidgetManager.getInstance(context);
```

步骤04 通过 RomoteViews 布局对 App Widget 进行更新。

```
    RemoteViews views = new RemoteViews(context.getPackageName(),
R.layout.example_appwidget);
appWidgetManager.updateAppWidget(mAppWidgetId, views);
```

步骤05 创建返回 Intent，设置 Activity 返回值，并关闭 Activity。

```
    Intent resultValue = new Intent();
resultValue.putExtra(AppWidgetManager.EXTRA_APPWIDGET_ID, mAppWidgetId);
setResult(RESULT_OK, resultValue);
finish();
```

3.6 进程和线程

当一个应用组件启动，并且该应用没有别的正在运行的组件时，则 Android 系统会为这个应用程序创建一个包含单个线程的 linux 进程。默认情况下，同一个应用程序的所有组件都运行在同一个进程与线程中（叫作“main”主线程）。某个应用组件启动，如果该应用程序的进程已经存在（因为应用程序的其他组件已经在运行了），那么刚刚启动的组件会在已有的进程和线程中启动运行。不过，可以指定组件运行在其他进程中，也可以为任何进程创建其他的线程。

本节主要讨论进程和线程是如何在 Android 应用程序中发挥作用的。

3.6.1 进程

默认情况下，同一个应用程序内的所有组件都是运行在同一个进程中的，大部分应用程序也不会去改变它。不过，如果需要指定某个特定组件所属的进程，那么可以利用 manifest 文件来达到目的。

manifest 文件中的每种组件元素（<activity>、<service>、<receiver>和<provider>）都支持 android:process 属性，用于指定组件所属运行的进程。设置此属性即可实现每个组件在各自的进程中运行，或者某几个组件共享一个进程而其他组件不可以参与。设置此属性也可以让来自于不同应用程序的组件运行在同一个进程中，实现多个应用程序共享同一个 Linux 的 user ID，并且提供相同的签名认证。

<application>元素也支持 android:process 属性，用于指定所有组件的默认值。

如果内存不足，且又有其他为用户提供更紧急服务的进程需要更多内存时，Android 可能会决定关闭一个进程。在此进程中运行着的应用程序组件也会因此被销毁。当需要再次工作时，会为这些组件重新创建一个进程。

在决定关闭哪个进程的时候，Android 系统会权衡它们对于用户的重要程度。比如，相对于一

个拥有可视 Activity 的进程，更有可能去关闭一个持有一组不再对用户可见的 Activity 的进程。也就是说，是否终止一个进程，取决于运行在此进程中组件的状态。终止进程的判定规则将在后续内容中讨论。（注：一个进程的关闭级别，按照该进程中最高的级别来定义，如该进程中有 Activity 和 Service，那么该进程的级别为 Service。）

Android 系统试图尽可能长时间地保持应用程序进程，但为了新建的或者更为重要的进程，总是需要清除旧的进程以回收内存。为了决定保留或终止哪个进程，根据进程内运行的组件及这些组件的状态，系统把每个进程都划入一个“importance hierarchy”中。重要性最低的进程首先会被清除，然后是其次低的进程，以此类推，这都是回收系统资源所必需的。

“importance hierarchy”共有 5 级，下面按照重要程度列出了各类进程（第一类进程是最重要的，将最后一个被终止）。

（1）前台进程（Foreground Process）

用户正在请求的进程。当以下任何一个条件成立时，该进程被认为是前台进程：

- 持有一个用户正在与之交互的 Activity（Activity 对象的 onResume()方法已被调用）。
- 持有一个服务（Service），且该服务已被绑定到一个正在与用户交互的 Activity 上。
- 持有一个服务，且该服务在前台运行，即该服务 startForground()调用。
- 持有一个服务，且该服务正在执行其生命周期的回调方法（onCreate()、onStart()、onDestroy()）。
- 持有一个 BroadcastReceiver，且其正在执行 onReceive()方法。

通常，在一个给定的时间内，只有很少的前台进程存在。当系统内存匮乏，以至于它们不能全部继续运行时，它们会依序被清除。通常，这时设备已经到了内存分页状态（memory paging state），清除那些前台进程以确保用户响应。

（2）可视进程（Visible Process）

一个可视进程是没有前台组件的，但仍会影响用户在屏幕上所见内容的进程。当以下任何一个条件成立时，该进程被认为是可视进程：

- 持有一个 Activity，且该 Activity 没有处于前台，但是对于用户而言它仍然可见（onPause()方法被调用）。这是可能发生的，例如，一个前台 Activity 启动了一个对话框，而之前的 Activity 还允许显示在后面。
- 持有一个服务（Service），且该服务被绑定到一个可视（或一个前台）Activity。

一个可视进程是极其重要的，除非无法维持所有前台进程同时运行了，它们是不会被终止的。

（3）服务进程（Service Process）

此进程运行着由 startService()方法启动的服务，它不会升级为上述两个级别。尽管服务进程不直接和用户所见内容关联，但它们通常在执行一些用户关心的操作（比如在后台播放音乐或从网络下载数据）。因此，除非内存不足以维持所有前台、可视进程同时运行，系统会保持服务进程的运行。

（4）后台进程（Background Process）

一个后台进程持有一个对用户不可见的 Activity（Activity 对象的 onStop()方法已被调用）。这些进程对用户体验没有直接的影响，系统可能在任意时间终止它们，以回收内存供前台进程、可视

进程及服务进程使用。通常会有许多后台进程运行，所以它们被保存在一个 LRU（Least Recently Used）列表中，以确保最近被用户使用的 Activity 最后一个被终止。如果一个 Activity 正确实现了生命周期方法，并保存了当前的状态，则终止此类进程不会对用户体验产生显著的影响。因为当用户回到这个 Activity，这个 Activity 会恢复它所有可视的状态。关于保存和恢复状态的详细信息，请参阅 Activities 文档。

（5）空进程（Empty Process）

空进程不含任何活动应用程序组件。保留这种进程的唯一目的就是用作缓存，以改善下次在此进程中运行组件的启动时间。为了在进程缓存和内核缓存间平衡系统整体资源，系统经常会终止这种进程。

依据进程中目前活跃组件的重要程度，Android 会给进程评估一个尽可能高的等级。例如，一个进程拥有一个服务和一个用户可见的 Activity，则此进程会被评定为可视进程，而不是服务进程。

此外，一个进程的等级可能会由于其他进程的依赖而被提高，一个服务于另一个进程的进程永远不能比另一个进程的等级低。比如，进程 A 中的 content provider 为进程 B 中的客户端提供服务，或进程 A 中的服务被进程 B 中的组件所调用，则进程 A 被认为其重要等级不低于进程 B。

因为运行服务的进程级别高于后台 Activity 进程的等级，所以，如果 Activity 需要启动一个长时间运行的操作，则为其启动一个服务（Service）会比简单地创建一个工作线程更好些，尤其是在此操作时间比 Activity 本身存在时间还要长久的情况下。比如，一个 Activity 要把图片上传至 Web 网站，就应该创建一个服务来执行，即使用户离开此 Activity，上传还是会在后台继续运行。无论 Activity 发生什么情况，使用服务可以保证操作至少拥有服务进程（service process）的优先级。同理，前面的广播接收器也是使用服务而非简单地启用一个线程。

3.6.2　线程

应用程序启动时，系统会为它创建一个名为“main”的主线程。主线程非常重要，因为它负责分配事件到合适的用户接口，包括绘图事件。它也是应用程序与 Android UI 组件包（来自 android.widget 和 android.view 包）进行交互的线程。因此，主线程有时也被叫作 UI 线程。

系统并不会为每个组件的实例创建单独的线程。运行于同一个进程中的所有组件都是在 UI 线程中实例化的，对每个组件的系统调用也都是由 UI 线程分配的。因此，对系统回调进行响应的方法（比如报告用户操作的 onKeyDown()或生命周期回调方法）总是运行在 UI 线程中。

例如，当用户触摸屏幕上的按钮时，应用程序的 UI 线程会把触摸事件分发给 widget，widget 先把自己置为按下（pressed）状态，再发送一个显示区域已失效（invalidate）的请求到事件队列中。UI 线程从队列中取出此请求，并通知 widget 重绘自己。

如果应用程序在与用户交互的同时需要执行繁重密集的任务，单线程模式可能会导致运行性能很低下，除非应用程序的执行时机很合适。如果 UI 线程需要处理每一件事情，那些耗时很长的操作（诸如访问网络或查询数据库等）将会阻塞整个 UI（线程）。一旦线程被阻塞，所有事件都不能被分发，包括屏幕绘图事件。从用户的角度来看，应用程序看上去似乎被挂起了。更糟糕的是，如果 UI 线程被阻塞超过一定时间（目前设置大约是 5 秒钟），用户就会被提示那个可恶的“应用程序没有响应”（ANR）。如果引起用户不满，可能就会决定退出并删除这个应用程序。

此外，Android 的 UI 组件包并不是线程安全的，因此不允许从工作线程中操作 UI，只能从 UI 线程中操作用户界面。因此，Android 的单线程模式必须遵守两个规则：

- 不允许阻塞 UI 线程。
- 不允许在 UI 线程之外访问 Android 的 UI 组件包。

根据以上对单线程模式的描述，要想保证程序界面的响应能力，关键是不能阻塞 UI 线程。如果操作不能很快完成，就让它们在单独的线程中运行（“后台”或“工作”线程）。

例如，以下是响应鼠标单击的代码，实现了在单独线程中下载图片并在 ImageView 显示的功能。

```
    public void onClick(View v) {
   new Thread(new Runnable() {
       public void run() {
           Bitmap b = loadImageFromNetwork("http://example.com/image.png");
           mImageView.setImageBitmap(b);
       }
   }).start();
}
```

首先，因为创建了一个新的线程来处理访问网络的操作，这段代码似乎能运行得很好。可是它违反了单线程模式的第二条规则，即不要在 UI 线程之外访问 Android 的 UI 组件包。这个例子在工作线程里而不是 UI 线程里修改了 ImageView，这可能导致不明确、不可预见的后果，要跟踪这种情况也是很困难很耗时的。

为了解决以上问题，Android 提供了几种方法，从其他线程中访问 UI 线程。下面列出了有助于解决问题的几种方法：

- Activity.runOnUiThread(Runnable)
- View.post(Runnable)
- View.postDelayed(Runnable, long)

例如，可以使用 View.post(Runnable)方法来修正上面的代码：

```
    public void onClick(View v) {
   new Thread(new Runnable() {
       public void run() {
           final Bitmap bitmap =
                   loadImageFromNetwork("http://example.com/image.png");
           mImageView.post(new Runnable() {
               public void run() {
                   mImageView.setImageBitmap(bitmap);
               }
           });
       }
   }).start();
}
```

现在，这段代码的执行是线程安全的了。网络相关的操作在单独的线程里完成，而 ImageView 是在 UI 线程里操纵的。

不过，随着操作变得越来越复杂，这类代码也会变得复杂难以维护。为了用工作线程完成更

加复杂的交互处理，可以考虑在工作线程中用 Handler 来处理 UI 线程分发过来的消息。当然，最好的解决方案也许就是继承使用异步任务类 AsyncTask，此类简化了一些工作线程和 UI 交互的操作。

（2）使用异步任务（AsyncTask）

异步任务允许以异步的方式对用户界面进行操作。它先阻塞工作线程，再在 UI 线程中呈现结果，在此过程中不需要对线程和 Handler 进行人工干预。

要使用异步任务，必须继承 AsyncTask 类并实现 doInBackground()回调方法，该对象将运行于一个后台线程池中。要更新 UI 时，需实现 onPostExecute()方法来分发 doInBackground()返回的结果。由于此方法运行在 UI 线程中，因此能够安全地更新 UI。然后就可以在 UI 线程中调用 execute()来执行任务了。

例如，可以利用 AsyncTask 来实现上面的例子：

```
    public void onClick(View v) {
    new DownloadImageTask().execute("http://example.com/image.png");
}

private class DownloadImageTask extends AsyncTask<String, Void, Bitmap> {
    /** The system calls this to perform work in a worker thread and
      * delivers it the parameters given to AsyncTask.execute() */
    protected Bitmap doInBackground(String... urls) {
        return loadImageFromNetwork(urls[0]);
    }

    /** The system calls this to perform work in the UI thread and delivers
      * the result from doInBackground() */
    protected void onPostExecute(Bitmap result) {
        mImageView.setImageBitmap(result);
    }
}
```

现在的 UI 是安全的，代码也得到了简化，因为任务分解成了工作线程内完成的部分和 UI 线程内完成的部分。

要全面理解这个类的使用，需阅读 AsyncTask 的参考文档。以下是关于其工作方式的概述：

- 可以用 generics 来指定参数的类型、进度值和任务最终值。
- 工作线程中的 doInBackground()方法会自动执行。
- onPreExecute()、onPostExecute()和 onProgressUpdate()方法都在 UI 线程中调用。
- doInBackground()的返回值会传给 onPostExecute()。
- 在 doInBackground()内的任何时刻，都可以调用 publishProgress()来执行 UI 线程中的 onProgressUpdate()。
- 可以在任何时刻、任何线程内取消任务。

注　意

在使用工作线程时，可能遇到的另一个问题是，由于运行配置的改变（比如用户改变了屏幕方向）导致 Activity 意外重启，这可能会销毁该工作线程。要了解如何在这种情况下维持任务执行以及如何在 Activity 被销毁时正确地取消任务，请参见 Shelves 例程的源代码。

3.6.3 线程安全方法

在一些情况下，实现的方法可能会被多个线程调用，因此应该设计为线程安全的。

真是存在能被远程调用的方法（比如，绑定服务（bound service）中的方法），当一个方法（在一个 IBinder 中实现）的调用发起于同一个进程（IBinder 正运行的）时，这个方法在调用者线程中执行。但是，如果调用发起于其他进程，那么这个方法将运行于线程池中选出的某个线程中（而不是运行于进程的 UI 线程中），该线程池由系统维护且位于 IBinder 所在的进程中。例如，即使一个服务的 onBind()方法是从服务所在进程的 UI 线程中调用的，实现了 onBind()的方法对象（比如，一个子类实现了 RPC 的方法）仍会从线程池中的线程被调用。因为一个服务可以有多个客户端，所以同时可以有多个线程池与同一个 IBinder 方法相关联。因此，IBinder 方法必须实现为线程安全的。

类似地，content provider 也能接收来自其他进程的数据请求。尽管 ContentResolver 类、ContentProvider 类隐藏了进程间通信管理的细节，ContentProvider 中响应请求的方法有：query()、insert()、delete()、update()和 getType()方法，这些方法都会从 ContentProvider 所在进程的线程池中被调用，而不是进程的 UI 线程。由于这些方法可能会从很多线程中同时被调用，因此它们也必须实现为线程安全的。

3.6.4 进程间的通信

Android 利用远程过程调用（Remote Procedure Call，RPC）提供了一种进程间通信（IPC）机制，通过这种机制，被 Activity 或其他应用程序组件调用的方法将（在其他进程中）被远程执行，而所有的结果将被返回给调用者。这就要求把方法调用及其数据分解到操作系统可以理解的程度，并将其从本地的进程和地址空间传输至远程的进程和地址空间，然后在远程进程中重新组装并执行这个调用。执行后的返回值将被反向传输回来。Android 提供了执行 IPC 事务所需的全部代码，因此只要关注定义和实现 RPC 编程接口即可。

要执行 IPC，应用程序必须用 bindService()绑定到服务上。详情请参阅服务 Services 开发文档。

3.7 小　　结

本章主要介绍了以下内容：

（1）Android 应用程序的基本组成，包括 Activity、Service、BroadcastReceiver、ContentProvider、Intent。

（2）Activity 的创建、生命周期以及之间数据传递的方法。

（3）Android 资源的创建以及使用，AndroidManifest.xml 定义应用程序及其组件的结构和源数据。

3.8　习　　题

1. 简述 Activity 的生命周期。
2. 比较 Activity 之间数据传递三种方法的优缺点。
3. 尝试创建自己的 Activity，并进行数据传递。
4. Android 应用程序的四大组件是什么？分别有什么作用？

第 4 章

Android GUI 开发

Android 系统提供了丰富的可视化界面组件，包括菜单、按钮、对话框等。Android 系统采用 Java 程序设计中的 UI 设计思想，其中包括事件处理机制及布局管理方式。Android 系统中的所有 UI 类都是建立在 View 和 ViewGroup 两个类的基础之上的，所有 View 的子类称为 Widget，所有 ViewGroup 的子类称为 Layout。本章将详细介绍 Android N 的基础功能单元——Activity 的用户界面 UI 设计、用户界面 UI 组件及其事件处理的相关知识。

4.1 View 和 ViewGroup

Activity 是 Android 应用程序与用户交互的接口，每一个屏幕视图都对应一个 Activity。其实 Activity 本身无法显示在屏幕上，其更像一个用于装载可显示组件的容器。这就好比一个 JSP 页面，它本身并没有显示出来任何东西，负责显示的是 JSP 页面内的各种 HTML 标签，而 JSP 页面好比一个容器，负责将这些表情装载到页面内。那么在 Android 应用程序里，谁才是真正负责显示的那部分呢？答案是 View 和 ViewGroup，其中 ViewGroup 是 View 的子类。

Android UI 界面是通过 View（视图）和 ViewGroup 及其派生类组合而成的。其中 View 是所有 UI 组件的基类，基本上所有的高级 UI 组件都是继承 View 类实现的，如 TextView（文本框）、Button、List、EditText（编辑框）、Checkbox 等。一个 View 在屏幕占据一块矩形区域，负责渲染这块矩形区域，也可以处理这块矩形区域发生的事件，并可以设置该区域是否可见以及获取焦点等。而 ViewGroup 是容纳这些组件的容器，其本身也是从 View 中派生出来的，它继承于 Android.view.View，功能就是装载和管理下一层的 View 对象或 ViewGroup 对象，也就是说它是一个容纳其他元素的容器，负责对添加进来的 View 和 ViewGroup 进行管理和布局。View 和 ViewGroup 的关系如图 4.1 所示。

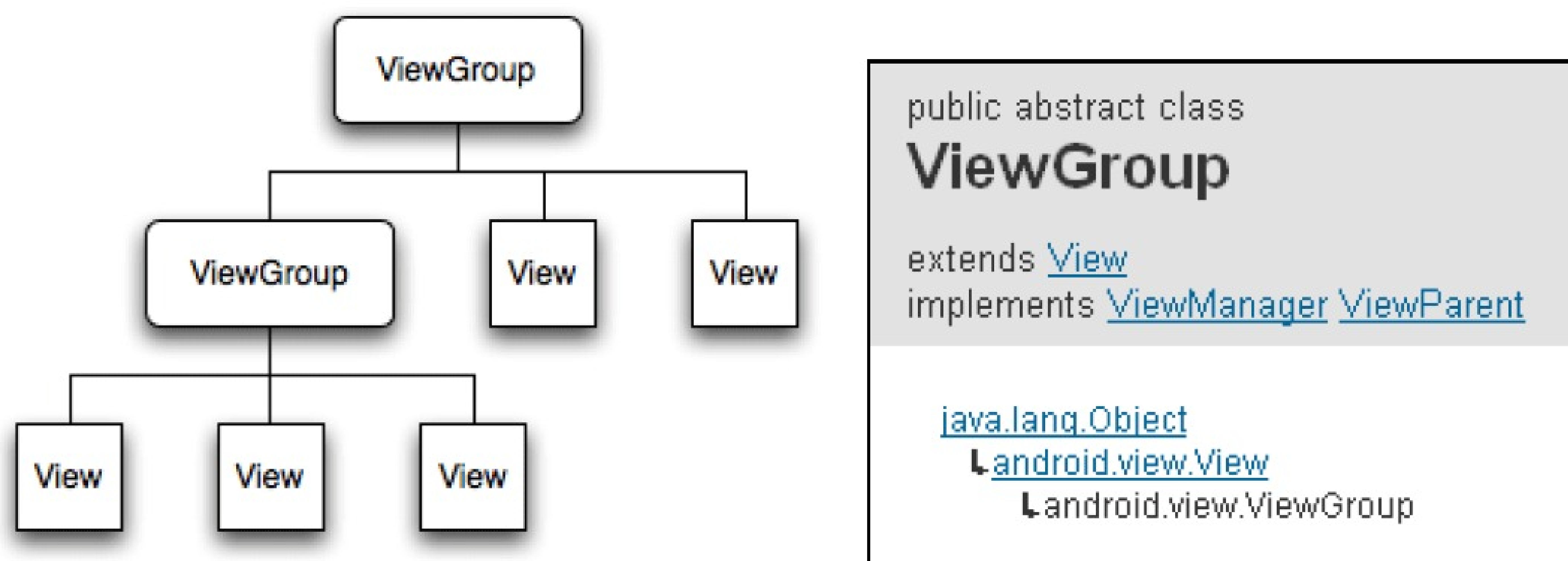


图 4.1 View 和 ViewGroup 的关系图

从图 4.1 可以看到，ViewGroup 可以包含一个或任意个 View（视图），也可以包含作为更低层次的子 ViewGroup，而子 ViewGroup 又可以包含下一层的叶子节点的 View 和 ViewGroup。这种灵活的层次关系可以形成复杂的 UI 布局。在开发过程中形成的用户界面 UI 一般来自于 View 和 ViewGroup 类的直接子类或者间接子类。

例如，View 派生出的直接子类有 AnalogClock、ImageView、KeyboardView、ProgressBar、Space、SurfaceView、TextView、TextureView、ViewGroup、ViewStub 等。ViewGroup 派生出的直接子类有 AbsoluteLayout、FragmentBreadCrumbs、FrameLayout、GridLayout、LinearLayout、RelativeLayout、SlidingDrawer 等。本章不能对 View 和 ViewGroup 的所有子类都进行详细的介绍，只能简单介绍其中常用的一小部分。如果需要了解各 UI 组件的相关信息，请参考相关文档。

4.2 使用 XML 定义视图

在使用 XML 构建一个用户界面之前，我们需要重温一下 Android 工程的目录结构。如图 4.2 所示，以 HelloAndroid 为例，project 视图列出了工程的目录结构。以.开头的目录是 AS 生成的辅助目录，无须用户干预。HelloAndroid 文件夹是模块目录，编程工作主要集中在这个目录中，相当于使用 Eclipse 构建的工程文件夹，包含 build、src、res 等文件夹。其中，res 目录为 Android 工程中所使用的资源目录，用户 UI 所涉及的资源基本都放置在该目录下。res 目录下的每一项资源文件都会由 AAPT（Android Asset Packaging Tool）为其生成一个对应的 public static final 类型的 ID 号，放置到 build 目录下的 R.java 文件中，Android 系统根据该 ID 号来访问对应

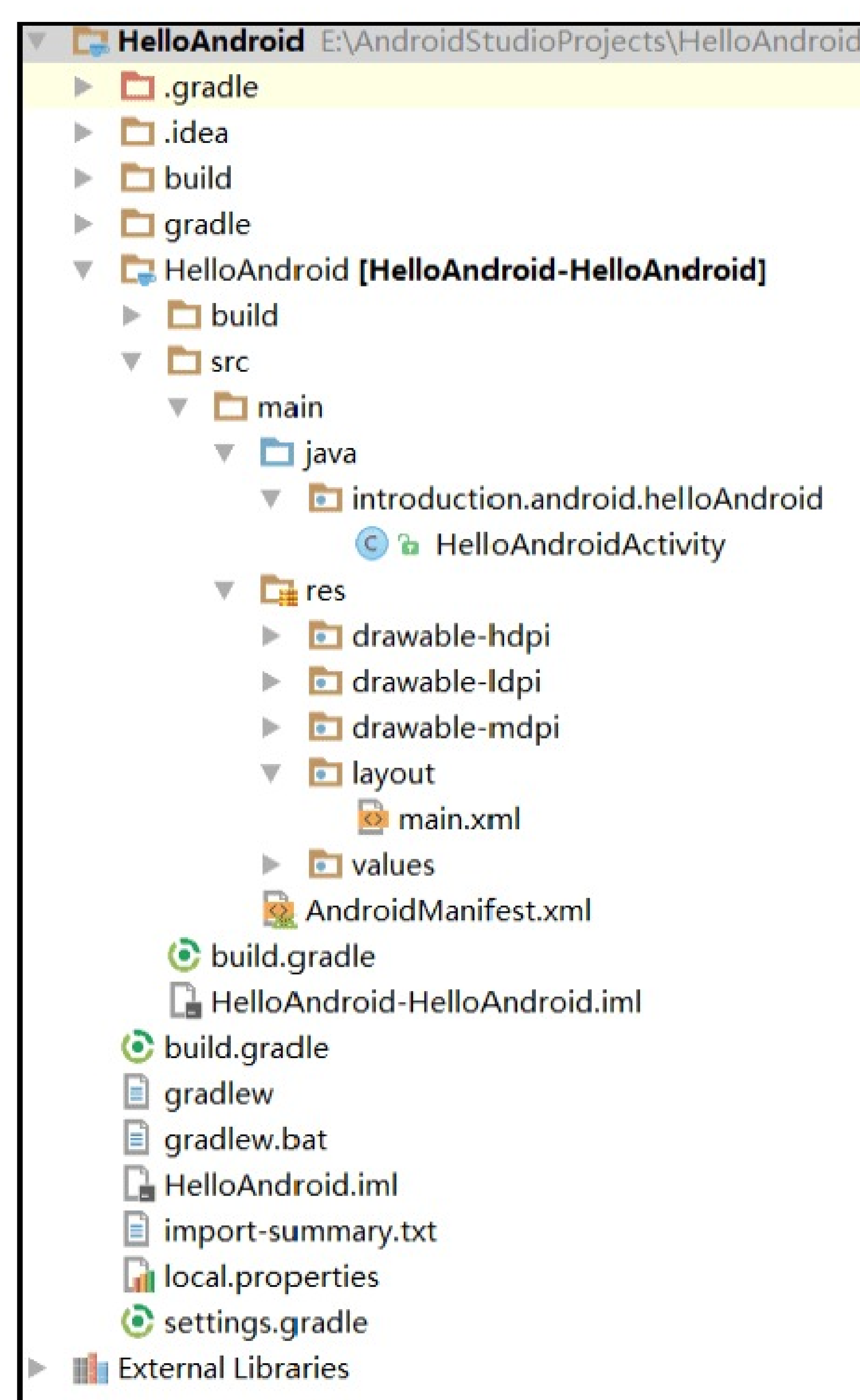


图 4.2 Android 工程的目录结构

资源。build 目录由 AS 自动生成，不需要用户修改，由系统维护。res/drawable/ 目录用来存放工程中使用到的图片文件，drawable 之后的 hdpi、ldpi、mdpi 分别放高分辨率、低分辨率和中分辨率的图片以适应不同分辨率的手机，Android 系统会根据用户手机的配置信息自动选取合适分辨率的图片文件，无须程序员干预；res/layout/目录下存放着定义 UI 布局文件用的 XML 文件，默认文件名为 main.xml；res/values/目录下存放着用于存储工程中所使用到的一些字符串信息的文件，默认文件名为 strings.xml。当然，每个目录下都可以存放多个 XML 文件，可由开发者自行创建。由此可见，Android 工程中使用的用户 UI 设计以及用户 UI 中涉及的字符串都是由 XML 文件来存储的。Android 系统使用 XML 文件来定义用户视图。

单击打开 values 文件夹下的 string.xml 文件显示出如下代码：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="hello">Hello Android!</string>
  <string name="app name">HelloAndroid</string>
</resources>
```

文件的开头部分<?xml version="1.0" encoding="utf-8"?>定义了 XML 的版本号和字符编码，这个部分在所有的 XML 文件中都会有，由系统自动添加，不需要修改。<resources>标签定义了 hello 和 app_name 两个变量，可以被 HelloAndroid 工程直接使用。当该文件被修改时，gen 目录下的 R.java 文件也会跟随进行更新。

双击 main.xml 文件，代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill parent"
   android:layout_height="fill parent"
   android:orientation="vertical">
<TextView
android:id="@id/textView1"
      android:layout_width="fill_parent"
      android:layout_height="wrap content"
      android:text="@string/hello" />
</LinearLayout>
```

<LinearLayout>标签定义了当前视图使用的是 LinearLayout 布局，也叫作线性布局，这是最常用的布局方式，Android SDK 还提供其他的几种布局方式，我们会在后面的章节中进行详细的介绍。在<LinearLayout>标签中定义了该布局方式的相关属性。android: layout_width="fill_parent" 和 android:layout_height="fill_parent" 表示该布局的宽和高充满整个手机屏幕，android:orientation="vertical"表示该布局中所放入的组件的排列方式为纵向排列。

在<LinearLayout ...>和</LinearLayout>之间可以添加各种 UI 组件并设置组件的相关属性，例如组件的高度、宽度、内容等，在 4.4 节会详细介绍各种常见组件的使用方法。在 HelloAndroid 实例中添加的是一个 TextView 组件，相当于一个显示内容的标签。android:layout_width="fill_parent" 指定其宽度覆盖满容器的宽，android:layout_height= "wrap_content"指定其高度跟随其显示内容变化。android:id="@id/textView1"指明该 TextView 的 ID 值为 R.java 文件中 ID 类的成员常量 textView1。Android SDK 提供了@[<package_ name>:]<resource_type>/<resource_name>方式，以便于从 XML 文件中访问工程的资源。android:text="@string/hello"指明该 TextView 组件显示的内容为

资源文件 string.xml 中定义的 hello 变量的内容。android:text 属性也可以直接指定要显示的字符串，但是在实际的工程开发过程中不鼓励这种方式，而应该使用资源文件中的变量，因为这样便于工程维护和国际化。在本书中，为了节省篇幅，部分显示内容简单的组件使用了字符串直接赋值的方法。

Android 工程中使用到的资源文件都会在 gen 目录下的 R.java 中生成对应项，由系统为每个资源分配一个十六进制的整型数值，唯一标明每个资源。

HelloAndroid 工程中的 R.java 文件代码如下：

```
package introduction.android.helloAndroid;

public final class R {
    public static final class attr {
    }
    public static final class drawable {
        public static final int ic launcher=0x7f020000;
}
public static final class id {
        public static final int textView1=0x7f050000;
    }
    public static final class layout {
        public static final int main=0x7f030000;
    }
    public static final class string {
        public static final int app_name=0x7f040001;
        public static final int hello=0x7f040000;
    }
}
```

由该文件可见，R 为静态最终类。其中 public static final class layout 代表的是 res/layout 文件夹的内容，layout 类的每个整型常量代表该文件夹下的一个 XML 布局文件。例如，public static final int main 代表的是 main.xml 文件，0x7f030000 为系统 main.xml 文件生成的整型数值。在 Android 工程中根据该数值找到 main.xml 文件。public static final class string 代表的是 res/values/strings.xml 文件，string 类中的每个整型常量型成员代表 strings.xml 文件中定义的一个变量。例如，public static final int app_name 代表 strings.xml 中定义的 app_name 变量，public static final int hello 代表 stings.xml 文件中定义的 hello 变量。

在工程开发过程中，可以通过[<package_name>.]R.<resource_type>.<resource_name>方式来访问 R 中定义的任意资源。其中 package_name 为资源文件被放置的包路径，一般可以省略。resource_type 为资源类型，例如 layout、string、color、drawable、menu 等。resource_ name 指的是为资源文件在类中定义的整型常量的名字，例如：

```
setContentView (R.layout.main) ;
```

这行代码中，通过 R.layout.main 找到了布局文件 main.xml，并通过 setContentView 方法将其设置为当前 Activity 的视图。要从视图中查找某个组件，需要使用 findViewById 方法，通过组件 ID 获取组件的对象。例如，要获取 main.xml 中的 TextView 组件对象，需要执行以下代码：

```
TextView textview= (TextView) findViewById (R.id.textView1) ;
```

4.3 布　　局

Android SDK 定义了多种布局方式以方便用户设计 UI。各种布局方式均为 ViewGroup 类的子类，结构如图 4.3 所示。本节将对 FrameLayout（单帧布局）、LinearLayout（线性布局）、AbsoluteLayout（绝对布局）、RelativeLayout（相对布局）和 TableLayout（表格布局）进行简单的介绍。

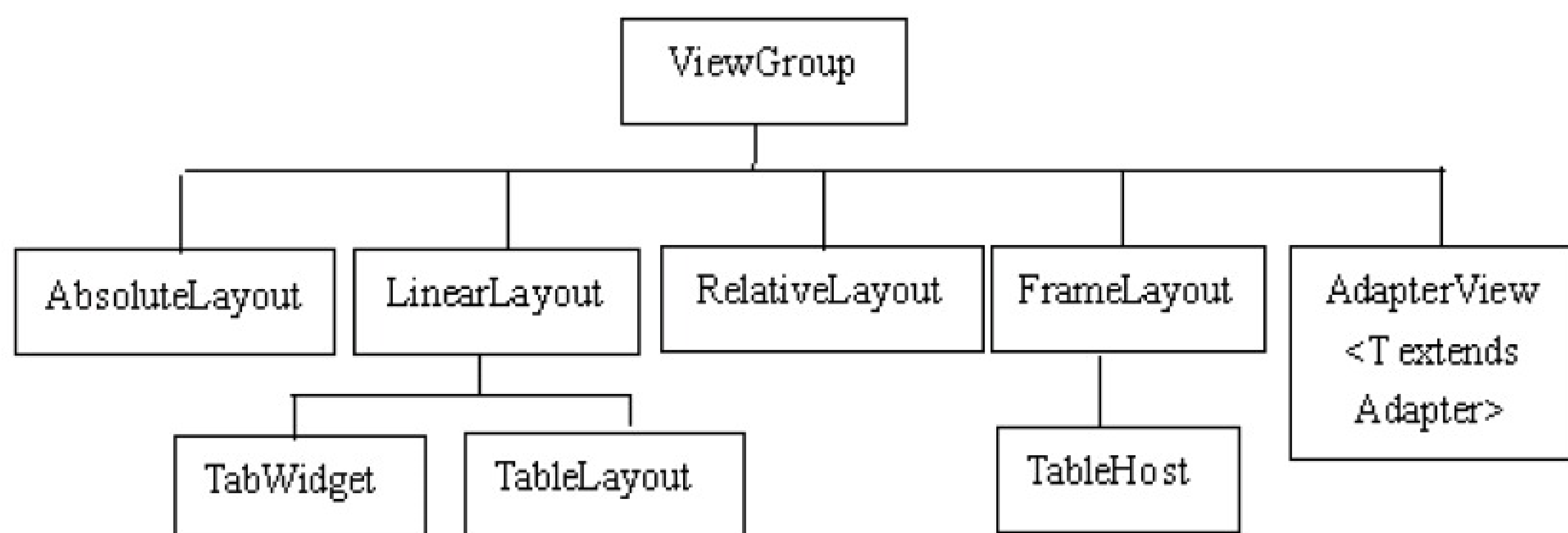


图 4.3　Android SDK 布局方式结构图

4.3.1　FrameLayout

FrameLayout 又称单帧布局，是 Android 所提供的布局方式里最简单的布局方式，它指定屏幕上的一块空白区域，在该区域填充一个单一对象，例如图片、文字、按钮等。应用程序开发人员不能为 FrameLayout 中填充的组件指定具体填充位置，默认情况下，这些组件都将被固定在屏幕的左上角，后放入的组件会放在前一个组件上进行覆盖填充，形成部分遮挡或全部遮挡。开发人员可以通过组件的 android:layout_gravity 属性对组件位置进行适当的修改。

实例 FrameLayoutDemo 演示了 FrameLayout 的布局效果。该布局中共有 4 个 TextView 组件，前 3 个组件以默认方式放置到布局中，第 4 个组件修改 gravity 属性后放置到布局中，运行效果如图 4.4 所示。



图 4.4　FrameLayout 的布局效果

实例 FrameLayoutDemo 中的布局文件 main.xml 的代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent">
    <TextView
        android:id="@+id/text1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

```
        android:textColor="#00ff00"
        android:textSize="100dip"
        android:text="@string/first"/>

     <TextView
        android:id="@+id/text2"
        android:layout_width="wrap content"
        android:layout_height="wrap content"
        android:textColor="#00ffff"
        android:textSize="70dip"
        android:text="@string/second"/>
   <TextView
        android:id="@+id/text3"
        android:layout_width="wrap content"
        android:layout_height="wrap content"
        android:textColor="#ff0000"
        android:textSize="40dip"
        android:text="@string/third"/>
   <TextView
        android:id="@+id/text4"
        android:layout_width="wrap content"
        android:layout_height="wrap_content"
        android:textColor="#ffff00"
        android:textSize="40dip"
        android:layout_gravity="bottom"
        android:text="@string/forth"/>
</FrameLayout>
```

其中：

```
android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

表明 FrameLayout 布局覆盖了整个屏幕空间。

实例 FrameLayoutDemo 中的 string.xml 文件内容如下：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
     <string name="app_name">FrameLayoutDemo</string>
   <string name="first">第一层</string>
   <string name="second">第二层</string>
   <string name="third">第三层</string>
   <string name="forth">第四层</string>
</resources>
```

从运行后的结果可见，前 3 个被放置到 FrameLayout 的 TextView 组件都是以屏幕左上角为基点进行叠加的。第 4 个 TextView 因为设置了 android:layout_gravity="bottom"属性而显示到了布局的下方。读者可自行将 android:layout_gravity 属性值修改为其他属性，查看运行效果。

4.3.2　LinearLayout

LinearLayout 又称线性布局，该布局应该是 Android 视图设计中最经常使用的布局。该布局可以使放入其中的组件以水平方式或者垂直方式整齐排列，通过 android:orientation 属性指定具体的

排列方式，通过 weight 属性设置每个组件在布局中所占的比重。

实例 LinearLayoutDemo 演示了 LinearLayout 布局的使用方法，效果如图 4.5 所示。

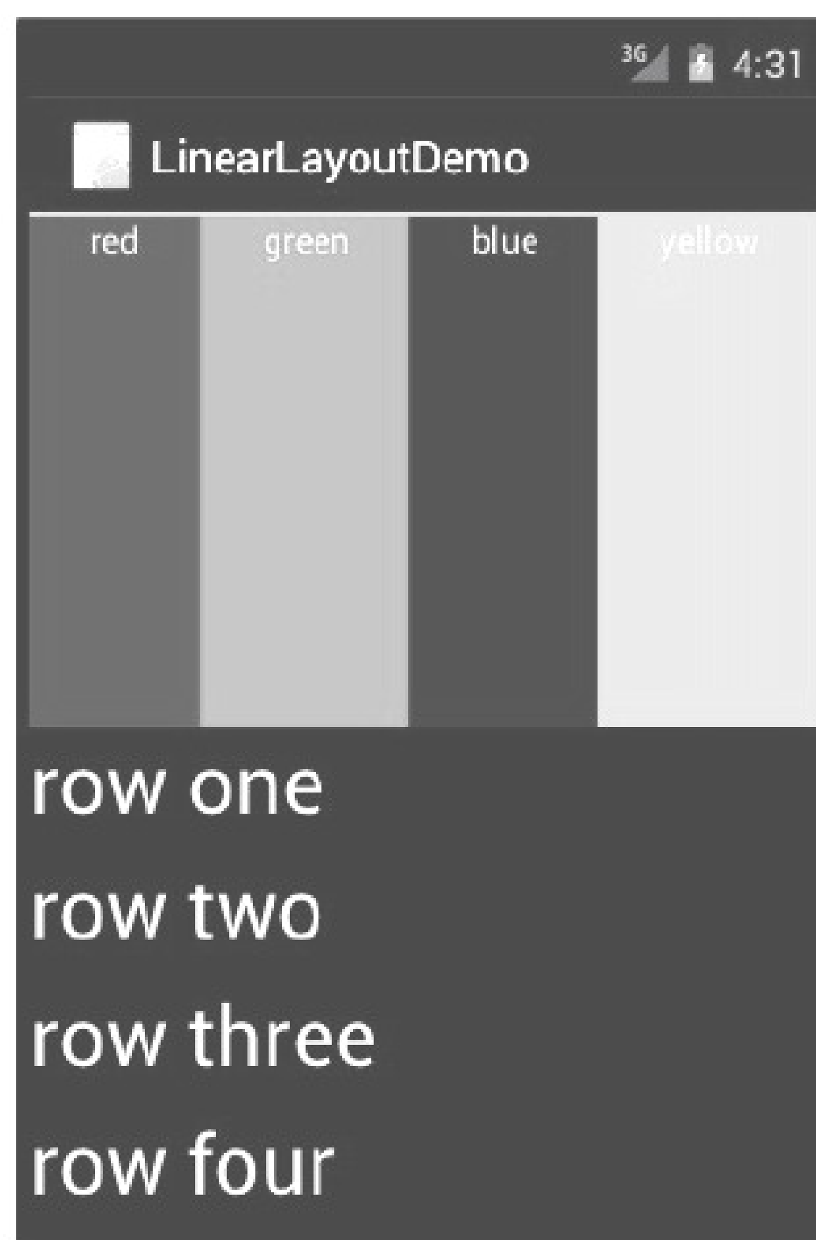


图 4.5　LinearLayout 的布局效果

实例 LinearLayoutDemo 中的 strings.xml 文件代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <string name="app_name">LinearLayoutDemo</string>
   <string name="red">red</string>
   <string name="yellow">yellow</string>
   <string name="green">green</string>
   <string name="blue">blue</string>
   <string name="row1">row one</string>
   <string name="row2">row two</string>
   <string name="row3">row three</string>
   <string name="row4">row four</string>
</resources>
```

实例 LinearLayoutDemo 中的布局文件 main.xml 代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">

 <LinearLayout
      android:orientation="horizontal"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:layout_weight="1">
     <TextView
          android:text="@string/red"
          android:gravity="center_horizontal"
          android:background="#aa0000"
          android:layout_width="wrap_content"
          android:layout_height="fill_parent"
```

```
            android:layout_weight="1"/>
        <TextView
            android:text="@string/green"
            android:gravity="center_horizontal"
            android:background="#00aa00"
            android:layout_width="wrap_content"
            android:layout_height="fill_parent"
            android:layout_weight="1"/>
        <TextView
            android:text="@string/blue"
            android:gravity="center_horizontal"
            android:background="#0000aa"
            android:layout_width="wrap_content"
            android:layout_height="fill_parent"
            android:layout_weight="1"/>
        <TextView
            android:text="@string/yellow"
            android:gravity="center_horizontal"
            android:background="#aaaa00"
            android:layout_width="wrap_content"
            android:layout_height="fill_parent"
            android:layout_weight="1"/>
</LinearLayout>

<LinearLayout
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:layout_weight="1">
  <TextView
      android:text="@string/row1"
      android:textSize="15pt"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:layout_weight="1"/>
  <TextView
      android:text="@string/row2"
      android:textSize="15pt"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:layout_weight="1"/>
  <TextView
      android:text="@string/row3"
      android:textSize="15pt"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:layout_weight="1"/>
  <TextView
      android:text="@string/row4"
      android:textSize="15pt"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:layout_weight="1"/>
</LinearLayout>
```

```
</LinearLayout>
```

该布局中放置了两个 LinearLayout 布局对象。第一个 LinearLayout 布局通过 android:orientation="horizontal" 属性将布局设置为横向线性排列，第二个 LinearLayout 布局通过 android:orientation="vertical"属性将布局设置为纵向线性排列。每个 LinearLayout 布局中都放入了 4 个 TextView，并通过 android:layout_weight 属性设置每个组件在布局中所占的比重相同，即各组件大小相同。layout_weight 用于定义一个线性布局中某组件的重要程度。所有的组件都有一个 layout_weight 值，默认为 0，意思是需要显示多大的视图就占据多大的屏幕空间。若赋值为大于 0 的值，则将可用的空间分割，分割的大小取决于当前的 layout_weight 数值与其他空间的 layout_weight 值的比率，例如水平方向上有两个按钮，每个按钮的 layout_weight 数值都设置为 1，那么这两个按钮平分宽度；若第一个为 1，第二个为 2，则可将空间的三分之一分给第一个，三分之二分给第二个。

将 LinearLayoutDemo 中水平 LinearLayout 的第 4 个 TextView 的 android:layout_weight 属性赋值为 2，运行效果如图 4.6 所示。

LinearLayout 布局可使用嵌套。活用 LinearLayout 布局可以设计出各种各样漂亮的布局方式。

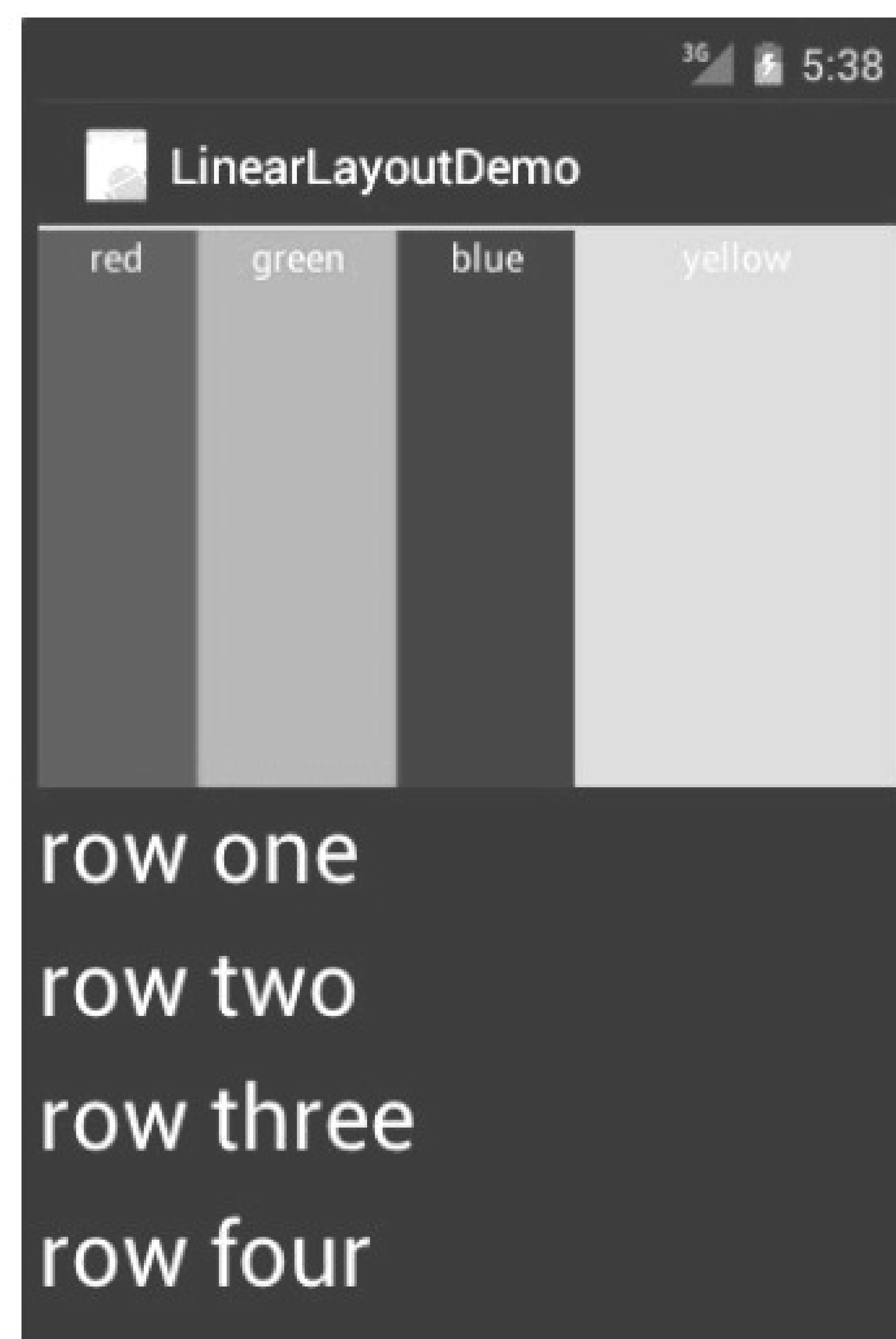


图 4.6　修改 android:layout_weight 属性

4.3.3　RelativeLayout

RelativeLayout 又称相对布局。从名称上可以看出，这种布局方式是以一种让组件以相对于容器或者相对于容器中的另一个组件的相对位置进行放置的布局方式。

RelativeLayout 布局提供了一些常用的布局设置属性用于确定组件在视图中的相对位置。相关属性及其所代表的含义列举在表 4.1 中。

表 4.1　RelativeLayout 布局常用属性

属性	描述
android:layout_above	将该组件的底部置于给定 ID 的组件之上
android:layout_below	将该组件的底部置于给定 ID 的组件之下
android:layout_toLeftOf	将该组件的右边缘与给定 ID 的组件左边缘对齐
android:layout_toRightOf	将该组件的左边缘与给定 ID 的组件右边缘对齐
android:layout_alignBottom	将该组件的底边与给定 ID 的组件底边对齐
android:layout_alignBaseline	将该组件的 baseline 与给定 ID 的 baseline 对齐
android:layout_alignTop	将该组件的顶部边缘与给定 ID 的顶部边缘对齐
android:layout_alignBottom	将该组件的底部边缘与给定 ID 的底部边缘对齐
android:layout_alignLeft	将该组件的左边缘与给定 ID 的左边缘对齐
android:layout_alignRight	将该组件的右边缘与给定 ID 的右边缘对齐
android:layout_alignParentTop	为 true，将该组件的顶部与其父组件的顶部对齐
android:layout_alignParentBottom	为 true，将该组件的底部与其父组件的底部对齐

（续表）

属性	描述
android:layout_alignParentLeft	为 true，将该组件的左侧与其父组件的左侧对齐
android:layout_alignParentRight	为 true，将该组件的右侧与其父组件的右侧对齐
android:layout_centerHorizontal	为 true，将该组件置于水平居中
android:layout_centerVertical	为 true，将该组件置于垂直居中
android:layout_centerInParent	为 true，将该组件置于父组件的中央

实例 RelativeLayoutDemo 演示了相对布局的使用方法，其运行效果如图 4.7 所示。



图 4.7　RelativeLayout 布局效果

实例 RelativeLayoutDemo 中的布局文件 main.xml 代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android="http://schemas.android.com/apk/res/android">

   <TextView
        android:id="@+id/label"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello" />

   <EditText
        android:id="@+id/enter"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_below="@+id/label" />

   <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentRight="true"
        android:layout_below="@+id/enter"
        android:text="@string/but1text" />
```

```
    <Button
        android:id="@+id/ok"
        android:layout_width="wrap content"
        android:layout_height="wrap content"
        android:layout_alignBottom="@+id/button1"
        android:layout_alignParentLeft="true"
        android:text="@string/but2text" />

</RelativeLayout>
```

该 RelativeLayout 布局的过程如下：

步骤01 **放置一个 ID 为 label 的 TextView 组件。**

步骤02 **通过 android:layout_below="@+id/label"属性将 ID 为 enter 的组件 EditText 放置到 TextView 的下面。**

步骤03 **在布局中加入一个 ID 为 button1 的 Button，通过 android:layout_below="@+ id/enter"属性将该 Button 放置到 enter 的下面，通过 android:layout_alignParentRight= "true"属性将 Button 放置到相对布局的右侧。**

步骤04 **在相对布局中加入一个名为 ok 的 Button，通过 android:layout_alignBottom="@+id/button1"属性将该 Button 底边与 button1 对齐，通过 android:layout_alignParentLeft ="true"属性将该 Button 放置到布局的左边。**

4.3.4 TableLayout

TableLayout 又称为表格布局，以行列的方式管理组件。TableLayout 布局没有边框，可以由多个 TableRow 对象或者其他组件组成，每个 TableRow 可以由多个单元格组成，每个单元格是一个 View。TableRow 不需要设置宽度 layout_width 和高度 layout_height，其宽度一定是 match_parent，即自动填满父容器，高度一定为 wrap_content，即根据内容改变高度。但对于 TableRow 中的其他组件来说，是可以设置宽度和高度的，只是必须是 wrap_content 或者 fill_parent。

实例 TableLayoutDemo 演示了使用 TableLayout 制作 UI 的方法，效果如图 4.8 所示。



图 4.8 TableLayout 布局效果

实例 TableLayoutDemo 中 strings.xml 的代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="hello">Hello World, TableLayout!</string>
  <string name="app name">TableLayoutDemo</string>
    <string name="column1">第一行第一列</string>
    <string name="column2">第一行第二列</string>
    <string name="column3">第一行第三列</string>
    <string name="empty">最左面的可伸缩 TextView</string>
    <string name="row2column2">第二行第三列</string>
    <string name="row2column3">End</string>
```

```
    <string name="merger">合并三个单元格</string>
</resources>
```

实例 TableLayoutDemo 中的布局文件 main.xml 的代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill parent"
   android:layout_height="fill parent" >
  <TableRow>
     <TextView
         android:text="@string/column1" />
     <TextView
         android:text="@string/column2"  />
     <TextView
         android:text="@string/column3" />
  </TableRow>

  <TextView
      android:layout_height="wrap content"
      android:background="#fff000"
      android:text="单独的一个TextView"
      android:gravity="center"/>
    <TableRow>
     <Button
         android:text="@string/merger"
         android:layout_span="3"
         android:gravity="center horizontal"
         android:textColor="#f00"
          />
  </TableRow>

    <TextView
      android:layout_height="wrap content"
      android:background="#fa05"
      android:text="单独的一个TextView"/>
    <TableRow android:layout_height="wrap content">
      <TextView
          android:text="@string/empty" />
      <Button
          android:text="@string/row2column2" />
        <Button
           android:text="@string/row2column3" />
    </TableRow>
</TableLayout>
```

布局文件 main.xml 在 TableLayout 布局内添加了两个 TableRow 和两个 TextView，形成了如图 4.8 所示的效果。从运行效果看，第一行和第五行都没能完全显示。TableLayout 布局提供了几个特殊属性，可以实现以下特殊效果。

- android:shrinkColumns 属性：该属性用于设置可收缩的列。当可收缩的列太宽以至于布局内的其他列不能完全显示时，可收缩列会纵向延伸，压缩自己所占的空间，以便于其他列可以完全显示出来。android:shrinkColumns="1"表示将第 2 列设置为可收缩列，列数从 0 开始。

- android:stretchColumns 属性：该属性用于设置可伸展的列。可伸展的列会自动扩展长度以填满所有可用空间。android:stretchColumns="1"表示将第 2 列设置为可伸展的列。
- android:collapseColumns 属性：该属性用于设置隐藏列。android:collapseColumns="1"表示将第 2 列隐藏不显示。

在<TableLayout>标签添加属性 android:shrinkColumns="0"，再次运行，效果如图 4.9 所示，可以看出第一行和第五行都完全显示出来了。



图 4.9　完全显示的效果

4.3.5　AbsoluteLayout

AbsoluteLayout 又称绝对布局，放入该布局的组件需要通过 android:layout_x 和 android:layout_y 两个属性指定其准确的坐标值，并显示在屏幕上。理论上，AbsoluteLayout 布局可用以完成任何的布局设计，灵活性很大，但是在实际的工程应用中不提倡使用这种布局。因为使用这种布局不但需要精确计算每个组件的大小，增大运算量，而且当应用程序在不同屏幕尺寸的手机上运行时会产生不同效果。

实例 AbsoluteLayoutDemo 演示了 AbsoluteLayout 布局的使用方法，效果如图 4.10 所示。



图 4.10　AbsoluteLayout 布局效果

实例 AbsoluteLayoutDemo 的布局文件 main.xml 代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<AbsoluteLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">

   <TextView
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:layout_x="10px"
       android:layout_y="10px"
       android:text="@string/hello">
   </TextView>

   <TextView
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:layout_x="80px"
       android:layout_y="80px"
       android:text="@string/action">
   </TextView>

   <TextView
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:layout_x="150px"
       android:layout_y="150px"
       android:text="@string/hello">
   </TextView>

</AbsoluteLayout>
```

其中：

```
android:layout_x="80px"
       android:layout_y="80px"
```

表示将组件放置到以屏幕左上角为坐标原点的坐标系下，x 值为 80 像素、y 值为 80 像素的位置。

在这里简单介绍一下 Android 系统常用的尺寸类型的单位。

- 像素：缩写为 px。表示屏幕上的物理像素。
- 磅：points，缩写为 pt。1pt 等于 1 英寸的 1/72，常用于印刷业。
- 放大像素：sp。主要用于字体的显示，Android 默认使用 sp 作为字号单位。
- 密度独立像素：缩写为 dip 或 dp。该尺寸使用 160dp 的屏幕作为参考，然后用该屏幕映射到实际屏幕，在不同分辨率的屏幕上会有相应的缩放效果以适用于不同分辨率的屏幕。若用 px 的话，320px 占满 HVGA 的宽度，到 WVGA 上就只能占一半不到的屏幕了，那一定不是你想要的。
- 毫米：mm。

4.3.6 WebView

WebView 组件是 AbsoluteLayout 的子类，用于显示 Web 页面。借助于 WebView，可以方便地开发自己的网络浏览器。此处仅对 WebView 的基本用法进行介绍，在后面进行 Web App 的学习时会有更进一步的讲解。

创建工程 WebViewDemo，为其在 AndroidManifest.xml 文件中添加 Internet 访问权限：

```
<uses-permission android:name="android.permission.INTERNET" />
```

在布局文件 main.xml 中添加一个 WebView 组件。Main.xml 内容如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill parent"
    android:orientation="vertical">

   <WebView
        android:id="@+id/webView1"
        android:layout_width="match parent"
        android:layout_height="match parent" />

</LinearLayout>
```

实例 WebViewDemo 中的 Activity 文件 WebViewDemoActivity.java 代码如下：

```
package introduction.android.webView;

import android.app.Activity;
import android.os.Bundle;
import android.webkit.WebView;

public class WebViewDemoActivity extends Activity {
    private WebView webView;

     /** Called when the activity is first created. */
    @Override
    public void onCreate (Bundle savedInstanceState) {
        super.onCreate (savedInstanceState) ;
        setContentView (R.layout.main) ;
        webView= (WebView) findViewById (R.id.webView1) ;
        webView.getSettings().setJavaScriptEnabled (true) ;
        webView.loadUrl ("http://www.google.com") ;
    }
}
```

运行效果如图 4.11 所示。

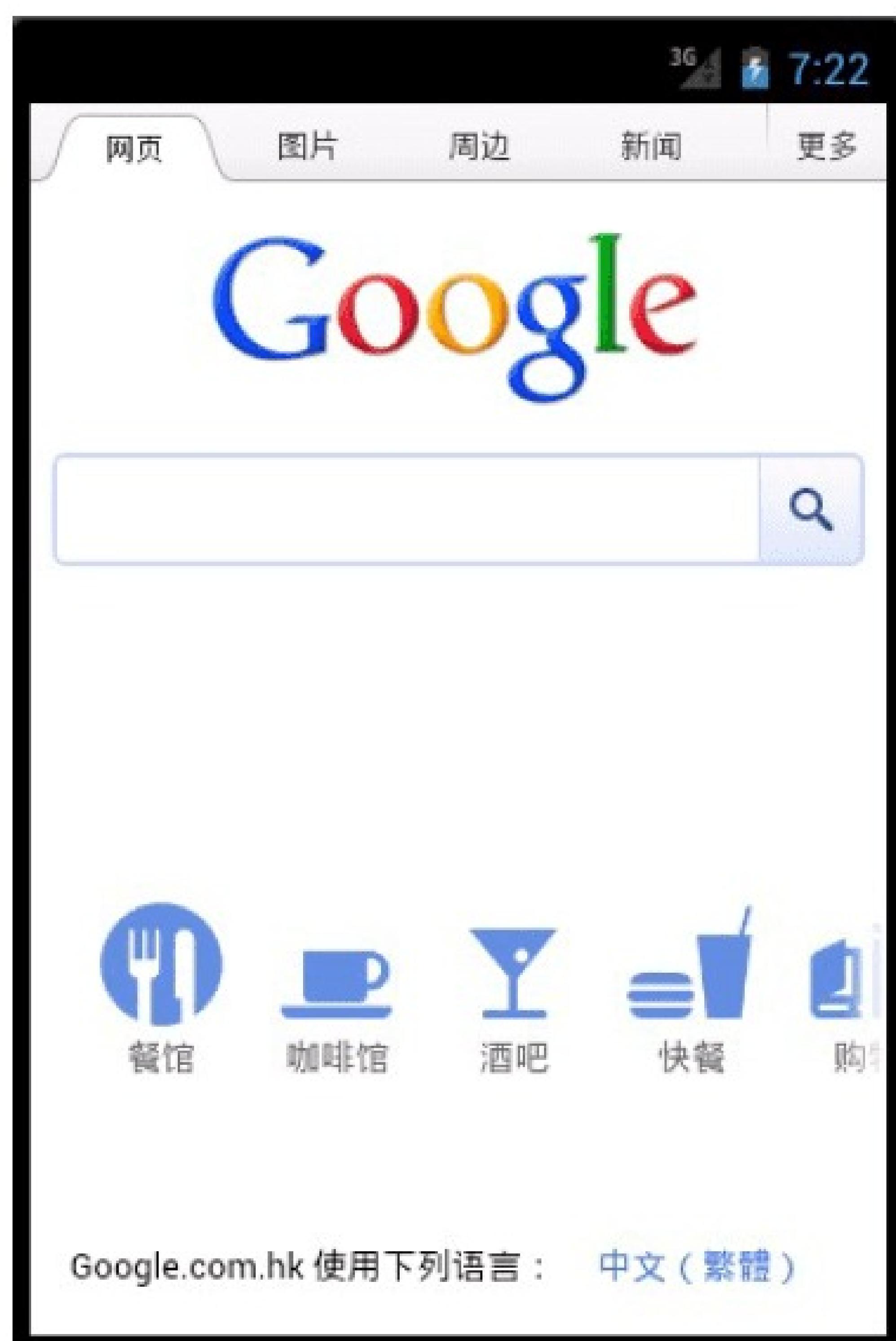


图 4.11　WebViewDemo 运行界面

4.4　常用 Widget 组件

在前面的章节讲解了用户界面 UI 设计中布局方面的知识，其中涉及少数几个常用的组件，例如按钮、文本框等。在这一节中着重讲解 Android 用户 UI 设计中常用的各种组件的用法。

Android SDK 提供了名为 android.widget 的包，其中提供了在应用程序界面设计中大部分常用的 UI 可视组件。之前章节中涉及的各种布局以及文本框、按钮等组件都包含在这个包中。Android 提供了强大的用户 UI 功能，要设计自己独特的应用程序界面，需要对各个组件有一个详细的了解。

4.4.1　创建 Widget 组件实例

在 Android Studio 中创建一个新的工程，名字为 WidgetDemo，用于对各种常见 UI 组件进行学习。下面是工程实现的步骤，在后续的章节中不会再赘述该过程。

步骤 01　新建项目。单击 File | New | New Project，打开 New Android Project 对话框，如图 4.12 所示。

步骤 02　输入工程名称 WidgetDemo，在 Location 后的文本框中输入工程的保存路径，单击 Next 按钮后，选择 API24:Android 7.0，再次单击 Next 按钮。

步骤 03　选择 EmptyActivity，确定 Activity 名字和 Layout 文件的名字，单击 Finish 按钮，则 AS 会生成工程目录和相关文件。若需要向以前版本兼容，则勾选“Backwards Compatibility(AppCompat)”复选框即可。本书中默认不勾选。

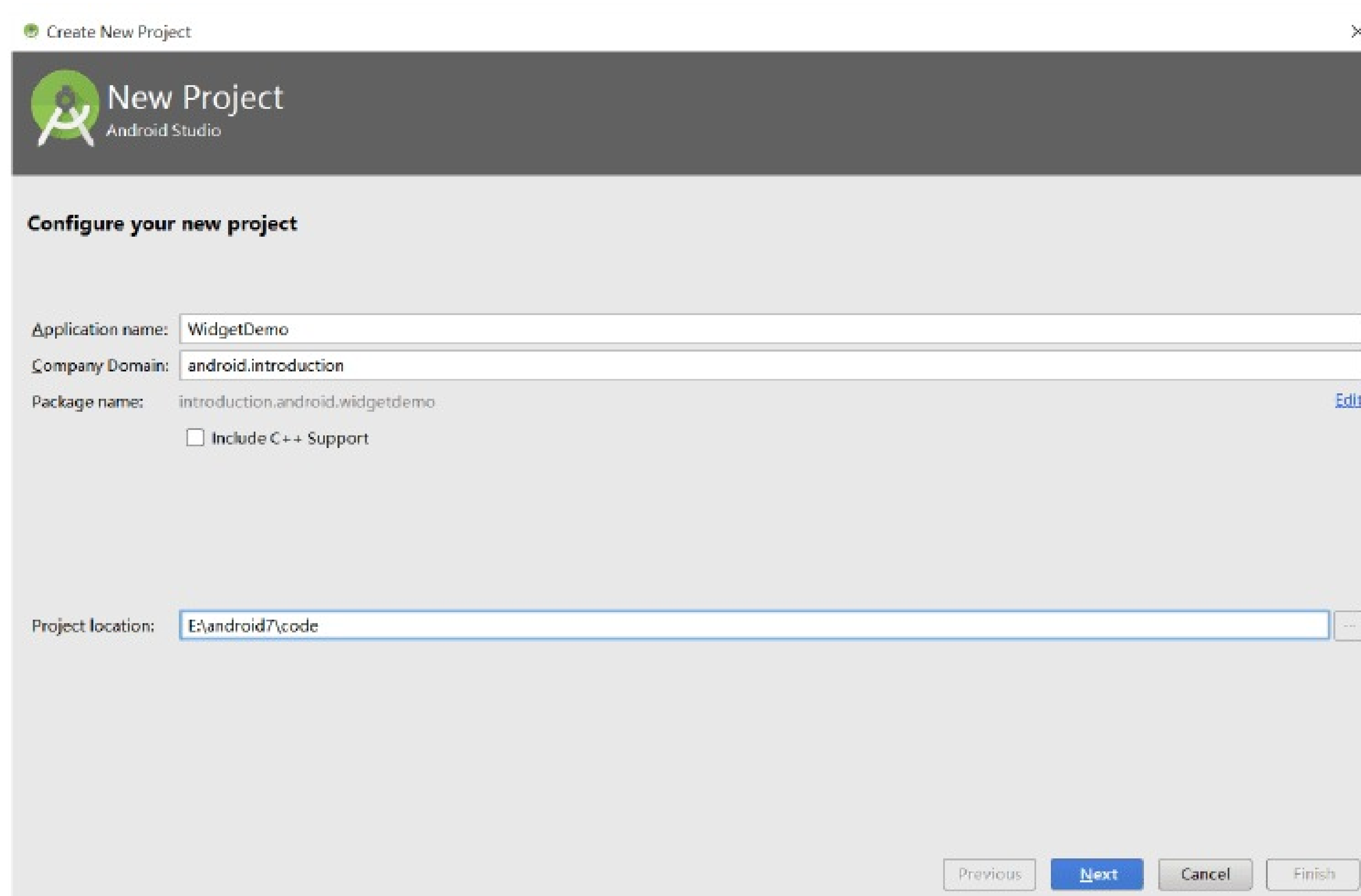


图 4.12　新建项目

WidgetDemoActivity.java 文件是当前应用程序的入口类 WidgetDemoActivity 的定义文件。双击 WidgetDemoActivity.java，发现已经为其生成代码如下：

```
package introduction.android.widgetDemo;
import android.app.Activity;
import android.os.Bundle;
public class WidgetDemoActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate (Bundle savedInstanceState) {
        super.onCreate (savedInstanceState) ;
        setContentView (R.layout.main) ;
    }
}
```

其中，onCreate()方法中的 setContentView（R.layout.main）表明 WidgetDemoActivity 使用的用户界面 UI 文件为 main.xml。

双击 main.xml 文件，发现提供了"Graphical Layout"和"main.xml"两种浏览方式。其中"Graphical Layout"方式为以图形方式浏览 main.xml 文件，其效果等同于 main.xml 在手机设备上运行的效果；"main.xml"方式为以代码方式浏览 main.xml 文件。这两种方式是等效的，都可以对 main.xml 文件进行编辑和查看。单击"main.xml"标签，发现已经为其生成代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill parent"
    android:orientation="vertical">
   <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap content"
        android:text="@string/hello" />
</LinearLayout>
```

该文件表明，当前 main.xml 文件所使用的布局为 LinearLayout 布局，该布局自动填满整个手

机屏幕。在该布局中，放置了一个 TextView 组件，该 TextView 显示的内容为"@string/hello"，表示 string.xml 文件中定义的 hello 变量的内容。双击 values 目录下的 string.xml 文件，会发现 hello 变量对应的值为“Hello World, WidgetDemoActivity!”。

单击 main.xml 的“Graphical Layout”浏览方式，可查看当前文件的图形化效果，如图 4.13 所示。

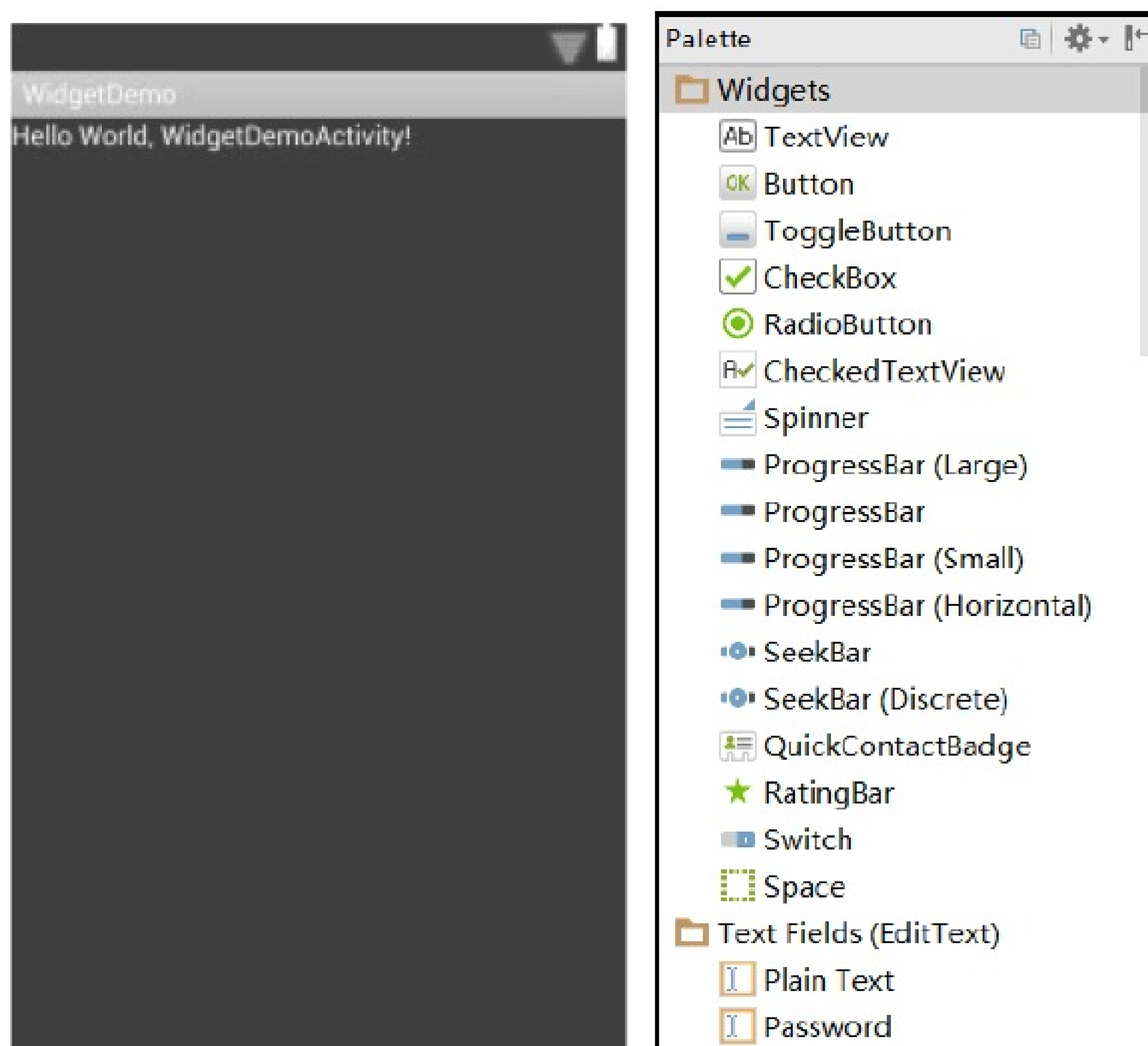


图 4.13　文件的图形化效果

程序开发人员可以在该图形方式下将左侧的各种组件直接拖曳到屏幕上，形成自己想要的布局，也可以直接修改 main.xml 文件的代码。

在后续章节中，在对布局文件进行修改时，若非特殊情况，将不再单独描述。

4.4.2　按钮

按钮（Button）应该是用户交互中使用最多的组件，在很多应用程序中都很常见。当用户单击按钮的时候，会有相对应的响应动作。下面在 WidgetDemo 工程的主界面 main.xml 中放置一个名为 Button 的按钮。文件代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical">
   <TextView
       android:layout_width="fill_parent"
       android:layout_height="wrap_content"
       android:text="@string/hello" />

   <Button
       android:id="@+id/button1"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:text="Button" />
```

```
</LinearLayout>
```

其中：

```
<Button
    android:id="@+id/button1"
    android:layout_width="wrap content"
    android:layout_height="wrap content"
    android:text="Button" />
```

表明在用户界面上放置了一个 ID 为“button1”的按钮，按钮的高度（layout_height）和宽度（layout_width）都会根据实际内容调整（wrap_content），按钮上显示文字为 Button，其运行效果如图 4.14 所示。

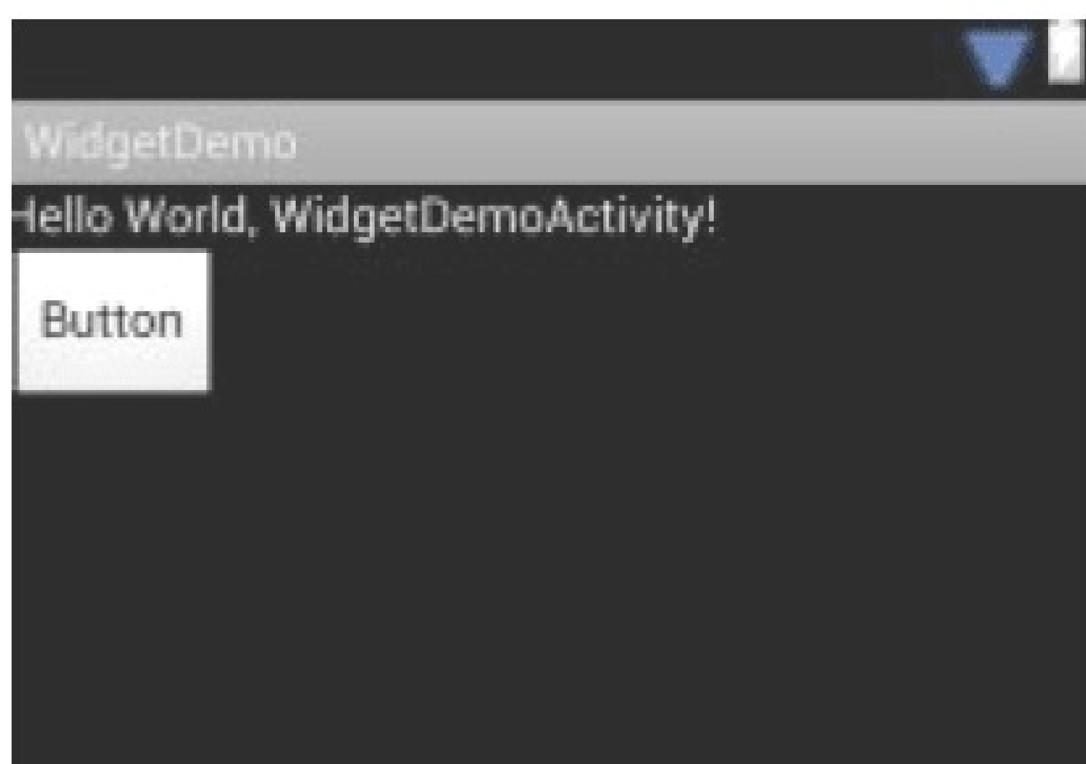


图 4.14　Button 的应用界面

按钮最重要的用户交互事件是“单击”事件。下面为 Button1 添加事件监听器和相应的单击事件。该过程在 WidgetDemoActivity.java 文件中完成，代码如下：

```
package introduction.android.widgetDemo;

import android.app.Activity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;

public class WidgetDemoActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate (Bundle savedInstanceState) {
        super.onCreate (savedInstanceState) ;
        setContentView (R.layout.main) ;
        Button btn= (Button) this.findViewById (R.id.button1) ;
        btn.setOnClickListener (new OnClickListener() {
                @Override
                public void onClick (View v) {
                    // TODO Auto-generated method stub
setTitle ("button1 被用户点击了") ;
                    Log.i ("widgetDemo", "button1 被用户点击了。") ;
                }
        }) ;
    }
```

```
    }
```

在 WidgetDemoActivity 的 onCreate()方法中，通过 findViewById(R.id.button1)方法获得 Button1 的对象，通过 setOnClickListener()方法为 Button1 设置监听器。此处新建了一个实现 OnClickListener 接口的匿名类作为监听器，并实现了 onClick()方法。当 Button1 被点击时，当前应用程序的标题被设置成“button1 被用户点击了”，对应在 LogCat 中也会打印相应的字符串，运行结果如图 4.15 所示。



图 4.15　点击按钮运行效果

4.4.3　文本框

文本框（TextView）是用于在界面上显示文字的组件，其显示的文本不可被用户直接编辑。程序开发人员可以设置 TextView 的字体大小、颜色、样式等属性。在工程 WidgetDemo 的 main.xml 中添加一个 TextView，代码如下：

```
<TextView
        android:id="@+id/textView1"
        android:layout_width="wrap content"
        android:layout_height="wrap_content"
        android:text="TextView" />
```

运行效果如图 4.16 所示。

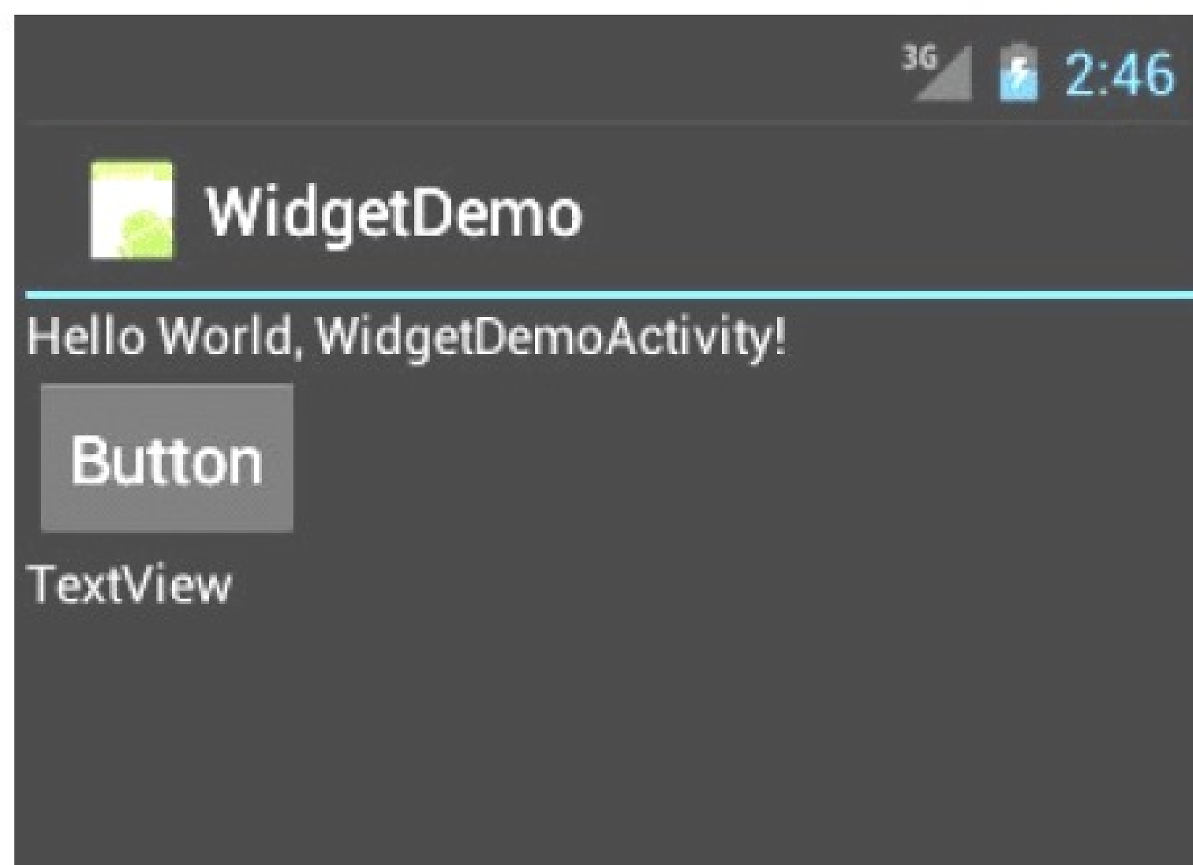


图 4.16　TextView 的应用界面

修改 Button1 的单击事件为：

```
public void onClick (View view) {
        // TODO Auto-generated method stub
        //setTitle ("button1 被用户单击了") ;
```

```
            Log.i ("widgetDemo", "button1 被用户单击了。") ;
    TextView textview= (TextView) findViewById (R.id.textView1) ;
            textview.setText ("设置 TextView 的字体") ;
            textview.setTextColor (Color.RED) ;
            textview.setTextSize (TypedValue.COMPLEX_UNIT_SP, 20) ;
    textview.setTypeface (Typeface.defaultFromStyle (Typeface.BOLD)) ;
        }
```

当 Button1 被单击时，通过 setText()方法更改 textView 的显示内容为“设置 TextView 的字体”，通过 setTextColor()方法修改 textView 显示字体的颜色为红色，通过 setTextSize()方法修改 textView 显示字体的大小为 20sp，通过 setTypeface()方法修改 textView 显示字体的风格为加粗，如图 4.17 所示。

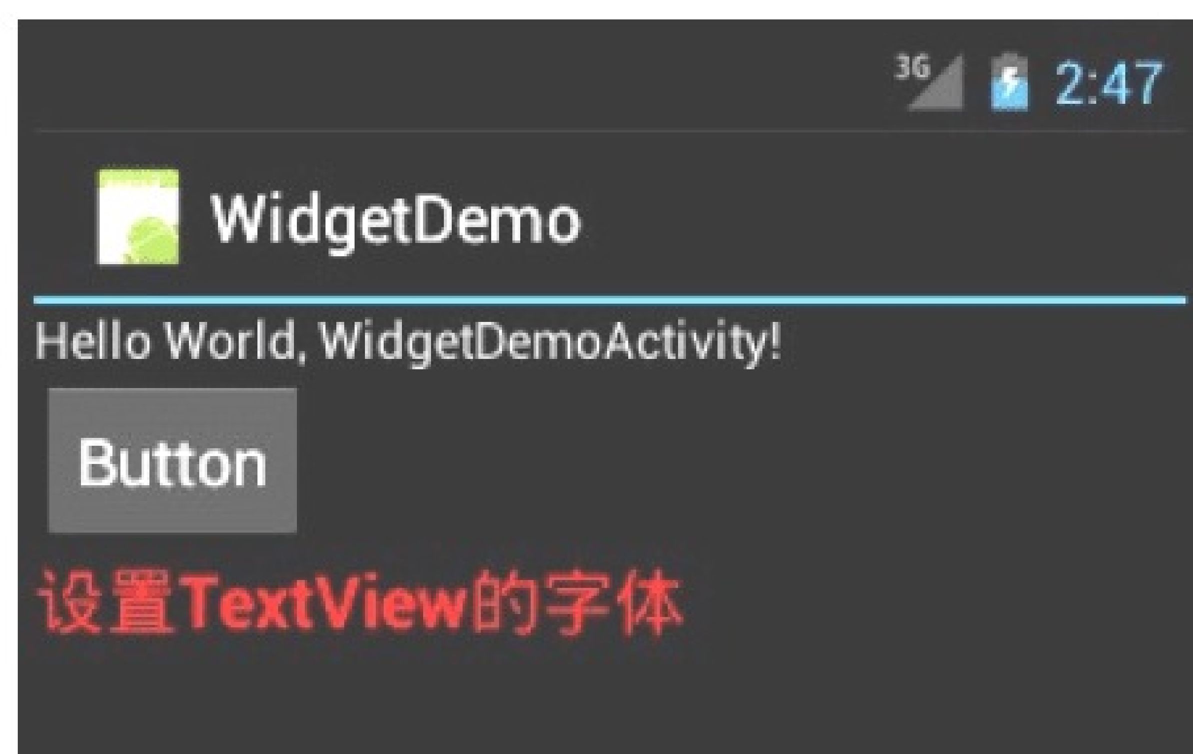


图 4.17　单击按钮运行效果

当然，该过程也可以通过修改 main.xml 文件来实现。将 TextView 标签按照如下代码修改也可以得到同样的效果，但是失去了应用程序中与用户交互的过程：

```
<TextView
    android:id="@+id/textView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="设置 TextView 的字体"
    android:textColor="#ff0000"
    android:textSize="20sp"
    android:textStyle="bold"/>
```

4.4.4　编辑框

编辑框（EditText）是 TextView 的子类，在 TextView 的基础上增加了文本编辑功能，用于处理用户输入，例如登录框等，是非常常用的组件。

在工程 WidgetDemo 的 main.xml 文件中添加一个 EditText，并实现这个功能：用户在 EditText 中输入信息的同时，用一个 TextView 显示用户输入的信息。

工程 WidgetDemo 中的布局文件 main.xml 中增加的代码如下：

```
<EditText
    android:id="@+id/editText1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content">
```

在 WidgetDemoActivity 的 onCreate()方法中添加下列代码：

```
editText=(EditText)findViewById(R.id.editText1);
editText.addTextChangedListener(new TextWatcher(){

    @Override
    public void afterTextChanged(Editable s) {
        // TODO Auto-generated method stub

    }

    @Override
    public void beforeTextChanged(CharSequence s, int start, int count,
        int after) {
        // TODO Auto-generated method stub

    }

    @Override
    public void onTextChanged(CharSequence s, int start, int before,
        int count) {
        // TODO Auto-generated method stub
        String text=editText.getText().toString();
        textview.setText(text);
    }

});
```

运行结果如图 4.18 所示。

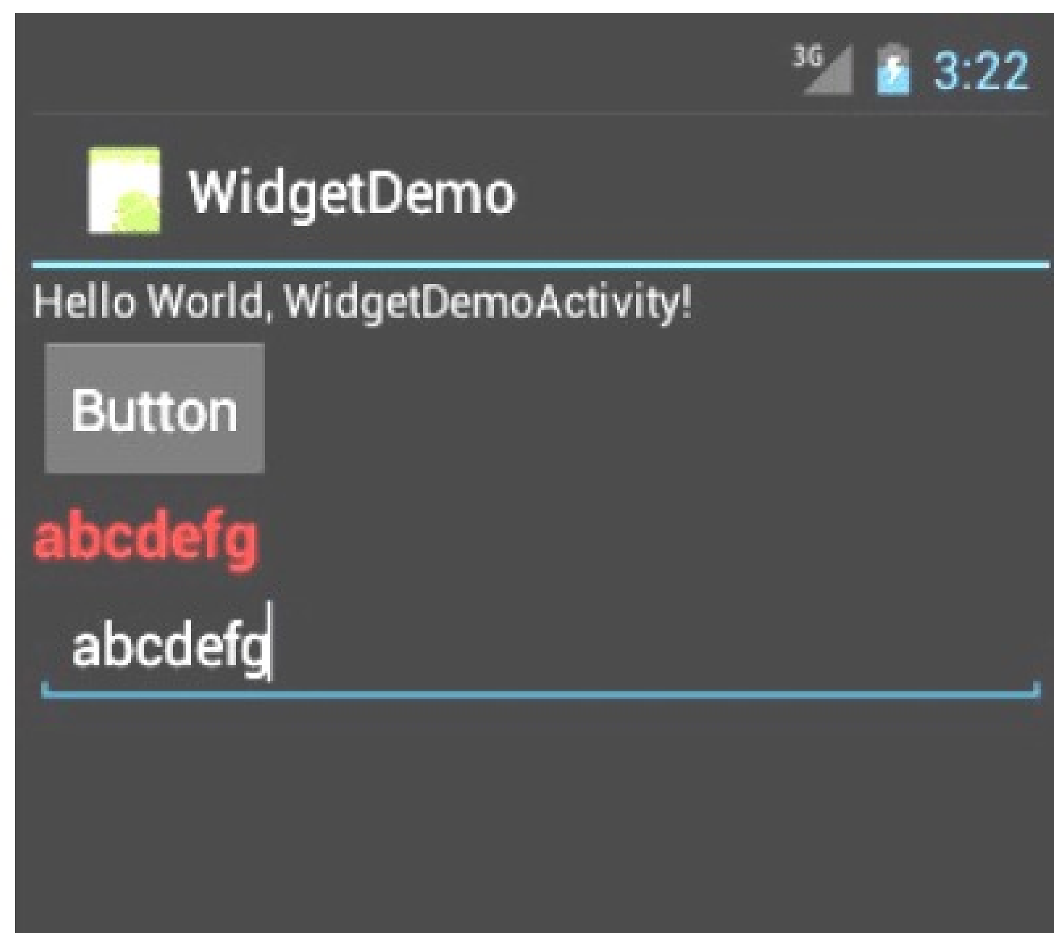


图 4.18　EditText 的应用界面

4.4.5　多项选择按钮

多项选择按钮（CheckBox）属于输入型组件，该组件允许用户一次选择多个选项。当用户不方便在手机屏幕上直接进行输入操作时，该组件的使用显得尤为方便。

下面通过实例讲解 CheckBox 的使用方法。该实例的运行效果如图 4.19 所示。



图 4.19　CheckBox 的应用界面

在工程 WidgetDemo 的布局文件 main.xml 文件中添加一个 Button，代码如下：

```
<Button
    android:id="@+id/button2"
    android:layout_width="wrap_content"
    android:layout_height="wrap content"
    android:text="CheckBoxDemo" />
```

当该 Button 被用户单击时，启动一个名为 CheckBoxActivity 的 Activity，在该 Activity 中演示 CheckBox 的使用方法。启动 CheckBoxActivity 的相关代码如下：

```
Button ckbtn=（Button）this.findViewById（R.id.button2）；
ckbtn.setOnClickListener（new OnClickListener(){

     @Override
     public void onClick（View v）{
          // TODO Auto-generated method stub
          Intent intent=new Intent（WidgetDemoActivity.this,CheckBoxActivity.class）；
          startActivity（intent）；
     }
})；
```

同时在 AndroidManifest.xml 文件中声明该 Activity：

```
<activity android:name="CheckBoxActivity"></activity>
```

CheckBoxActivity 所使用的布局文件为 checkbox.xml，使用 LinearLayout 布局，其中放置了一个 TextView 和三个 CheckBox。Checkbox.xml 的文件内容如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
```

```
    <TextView
        android:id="@+id/text"
        android:layout_width="fill parent"
        android:layout_height="wrap content"
        android:text="@string/checkboxhello"/>
    <CheckBox
        android:id="@+id/CheckBox1"
        android:layout_width="wrap content"
        android:layout_height="wrap_content"
        android:text="@string/football"/>
    <CheckBox
        android:id="@+id/CheckBox2"
        android:layout_width="wrap content"
        android:layout_height="wrap content"
        android:text="@string/song"/>
    <CheckBox
        android:id="@+id/CheckBox3"
        android:layout_width="wrap content"
        android:layout_height="wrap_content"
        android:text="@string/book"/>

</LinearLayout>
```

这 4 个组件在对应的 strings.xml 文件中定义的变量为：

```
<string name="checkboxhello">你的爱好是:</string>
  <string name="football">篮球</string>
  <string name="song">听歌曲</string>
<string name="book">看书</string>
```

当用户对多项选择按钮进行选择时，为了确定用户选择的是哪几项，需要对每个多项选择按钮进行监听。CompoundButton.OnCheckedChangedListener 接口可用于对 CheckBox 的状态进行监听。当 CheckBox 的状态在未被选中和被选中之间变化时，该接口的 onCheckedChanged()方法会被系统调用。CheckBox 通过 setOnCheckedChangeListener()方法将该接口对象设置为自己的监听器。

CheckBoxActivity.java 的代码如下：

```
package introduction.android.widgetDemo;

import android.app.Activity;
import android.os.Bundle;
import android.widget.CheckBox;
import android.widget.CompoundButton;
import android.widget.TextView;

public class CheckBoxActivity extends Activity {
    private TextView textView;
    private CheckBox bookCheckBox;
    private CheckBox songCheckBox;
    private CheckBox footbaCheckBox;
    @Override
    protected void onCreate (Bundle savedInstanceState) {
        // TODO Auto-generated method stub
        super.onCreate (savedInstanceState) ;
```

```
            this.setContentView(R.layout.checkbox);
            textView=(TextView)findViewById(R.id.text);
            footbaCheckBox=(CheckBox)findViewById(R.id.CheckBox1);
            songCheckBox=(CheckBox)findViewById(R.id.CheckBox2);
            bookCheckBox=(CheckBox)findViewById(R.id.CheckBox3);

            footbaCheckBox.setOnCheckedChangeListener(new
CompoundButton.OnCheckedChangeListener(){

            @Override
            public void onCheckedChanged(CompoundButton buttonView, boolean isChecked){
                // TODO Auto-generated method stub
                if(footbaCheckBox.isChecked()){
                   textView.append(footbaCheckBox.getText().toString());
                }else {
                   if(textView.getText().toString().contains("足球")){
                    textView.setText(textView.getText().toString().replace("足球", ""));
                   }
                }
              }
            });
            songCheckBox.setOnCheckedChangeListener(new
CompoundButton.OnCheckedChangeListener(){

               @Override
               public void onCheckedChanged(CompoundButton buttonView, boolean isChecked){
                // TODO Auto-generated method stub
                if(songCheckBox.isChecked()){
                   textView.append(songCheckBox.getText().toString());
                }else {
                   if(textView.getText().toString().contains("唱歌")){
                    textView.setText(textView.getText().toString().replace("唱歌", ""));
                   }
                }
              }
            });

            bookCheckBox.setOnCheckedChangeListener(new
CompoundButton.OnCheckedChangeListener(){

               @Override
               public void onCheckedChanged(CompoundButton buttonView, boolean isChecked){
                // TODO Auto-generated method stub
                if(bookCheckBox.isChecked()){
                   textView.append(bookCheckBox.getText().toString());
                }else {
                   if(textView.getText().toString().contains("读书")){
                    textView.setText(textView.getText().toString().replace("读书", ""));
                   }
                }
              }
            });
        }

    }
```

CheckBoxActivity 为 Checkbox.xml 文件中的三个 CheckBox 分别添加了监听器。当 CheckBox 的状态发生改变时，通过 Checkbox.isChecked()方法可以获取当前 CheckBox 按钮的选中状态，进而进行处理。

4.4.6　单项选择按钮组

RadioGroup 为单项选择按钮组，其中可以包含多个 RadioButton，即单选按钮，它们共同为用户提供一种多选一的选择方式。在多个 RadioButton 被同一个 RadioGroup 包含的情况下，多个 RadioButton 之间自动形成互斥关系，仅有一个可以被选择。单选按钮的使用方法和 CheckBox 的使用方法高度相似，其事件监听接口使用的是 RadioGroup.OnCheckedChangeListener()，使用 setOnCheckedChangeListener()方法将监听器设置到单选按钮上。按照 CheckBox 的讲解思路，启动一个名为 RadioGroupActivity 的 Activity 来对 RadioGroup 进行讲解。

RadioGroupActivity 的运行效果如图 4.20 所示。

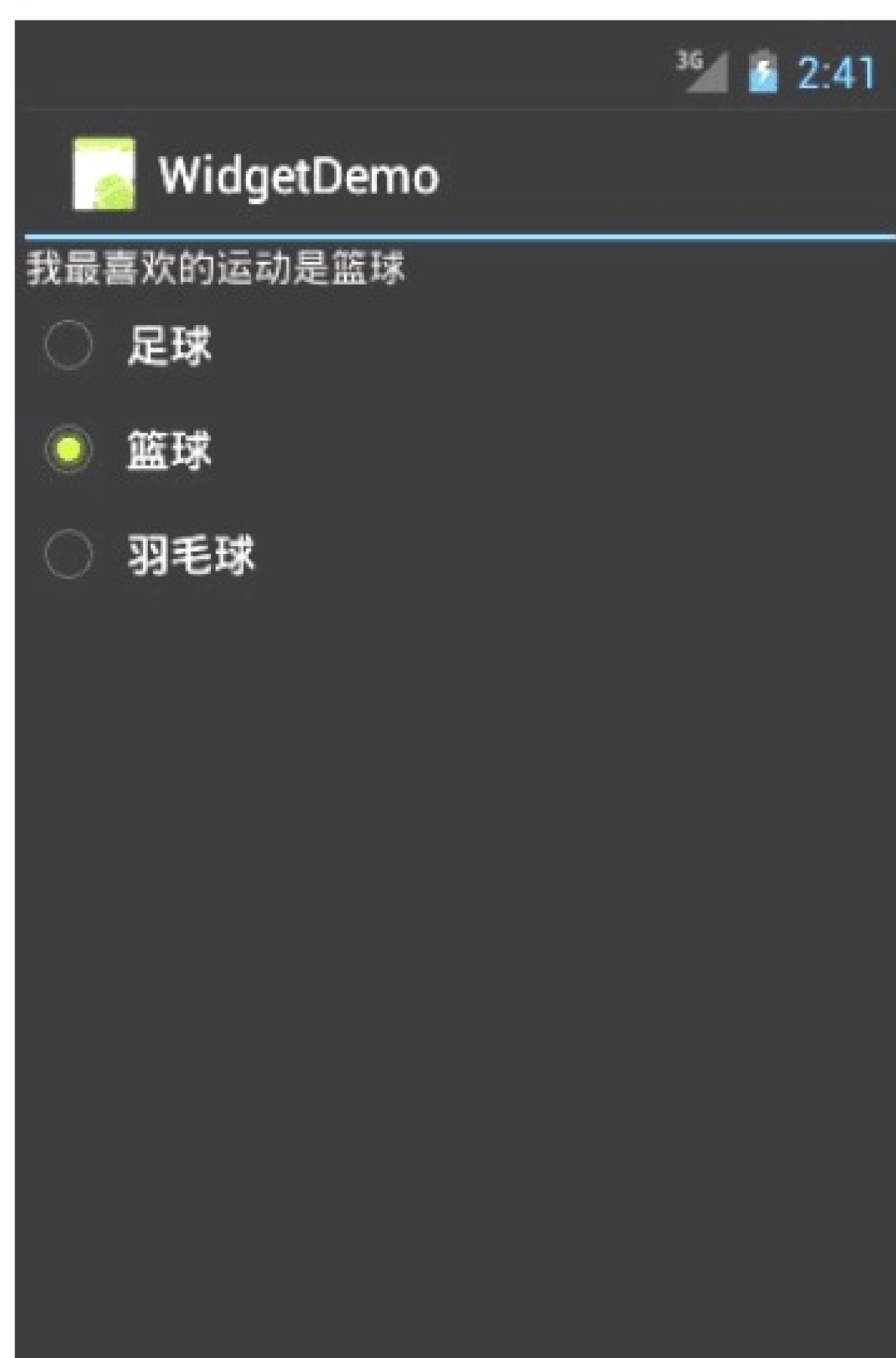


图 4.20　RadioGroup 的应用界面

在工程 WidgetDemo 的布局文件 main.xml 中添加一个 Button，并启动 RadioGroupActivity 的相关代码。在 main.xml 中添加代码如下：

```
<Button
    android:id="@+id/button3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="RadioGroupDemo" />
```

启动处理 RadioGroup 的 Activity RadioGroupActivity 的代码如下：

```
Button radiotn= (Button) this.findViewById (R.id.button3) ;
radiotn.setOnClickListener (new OnClickListener(){

    @Override
    public void onClick (View v) {
```

```
            // TODO Auto-generated method stub
            Intent intent=new Intent (WidgetDemoActivity.this,RadioGroupActivity.class) ;
            startActivity (intent) ;
        }
    }) ;
```

同时在 AndroidManifest.xml 文件中声明该 Activity：

```
<activity android:name=" RadioGroupActivity "></activity>
```

RadioGroupActivity 使用的是 radiogroup.xml，其代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match parent"
   android:layout_height="match_parent"
   android:orientation="vertical">

  <TextView
      android:id="@+id/radiohello"
      android:layout_width="fill parent"
      android:layout_height="wrap_content"
      android:text="@string/hello"/>

  <RadioGroup
      android:id="@+id/radiogroup1"
      android:layout_width="wrap content"
      android:layout_height="wrap_content"
      android:orientation="vertical"
      android:layout_x="3px"
  >
     <RadioButton
         android:id="@+id/radiobutton1"
         android:layout_width="wrap content"
         android:layout_height="wrap content"
         android:text="@string/football"
      />
     <RadioButton
         android:id="@+id/radiobutton2"
         android:layout_width="wrap content"
         android:layout_height="wrap content"
         android:text="@string/bascketball"
      />
     <RadioButton
         android:id="@+id/radiobutton3"
         android:layout_width="wrap content"
         android:layout_height="wrap_content"
         android:text="@string/badminton"
      />
  </RadioGroup>

</LinearLayout>
```

该布局文件使用了 LinearLayout 布局，并且在其中放置了一个 TextView 和一个 RadioGroup。RadioGroup 中含有三个 RadioButton。这些组件对应的 strings.xml 文件中定义的变量为：

```
<string name="radiohello">你最喜欢的运动是:</string>
  <string name="bascketball">篮球</string>
  <string name="badminton">羽毛球</string>
<string name="football">足球</string>
```

RadioGroupActivity.java 的代码如下：

```
package introduction.android.widgetDemo;

import android.app.Activity;
import android.os.Bundle;
import android.widget.RadioButton;
import android.widget.RadioGroup;
import android.widget.TextView;

public class RadioGroupActivity extends Activity {
    private TextView textview;
    private RadioGroup radiogroup;
    private RadioButton radio1,radio2,radio3;
    @Override
    public void onCreate (Bundle savedInstanceState) {
         super.onCreate (savedInstanceState) ;
         setContentView (R.layout.radiogroup) ;
         textview= (TextView) findViewById (R.id.radiohello) ;
         radiogroup= (RadioGroup) findViewById (R.id.radiogroup1) ;
         radio1= (RadioButton) findViewById (R.id.radiobutton1) ;
         radio2= (RadioButton) findViewById (R.id.radiobutton2) ;
         radio3= (RadioButton) findViewById (R.id.radiobutton3) ;
         radiogroup.setOnCheckedChangeListener (new RadioGroup.OnCheckedChangeListener(){

         @Override
         public void onCheckedChanged (RadioGroup group, int checkedId)
          {
              // TODO Auto-generated method stub
              String text="我最喜欢的运动是";
              if (checkedId==radio1.getId()) {
                   text+=radio1.getText().toString();
                 textview.setText (text) ;
              }else if (checkedId==radio2.getId()) {
                   text+=radio2.getText().toString();
                 textview.setText (text) ;
              }else if (checkedId==radio3.getId()) {
                   text+=radio3.getText().toString();
                 textview.setText (text) ;
              }
            }
         }) ;
    }
}
```

在 RadioGroupActivity 的 onCreate()方法中为 RadioGroup 添加监视器 RadioGroup。OnCheckedChangeListener 在其回调方法 onCheckedChanged()中对三个 RadioButton 分别进行处理。需要说明的是，如果把 RadioGroup 去掉，只使用 RadioButton 的话，则需要为每个 RadioButton 单独设置监听器，其使用方法和 CheckBox 没有任何区别。

4.4.7 下拉列表

Spinner 提供下拉列表式的输入方式，该方法可以有效节省手机屏幕上的显示空间。

下面用一个简单的实例讲解 Spinner 的使用方法。在工程 WidgetDemo 的布局文件 main.xml 中添加一个 Button，用以启动 SpinnerActivity。

在 main.xml 中添加代码如下：

```
<Button
        android:id="@+id/button4"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="SpinnerDemo" />
```

单击 Button 并启动 SpinnerActivity 的代码如下：

```
Button spinnerbtn= (Button) this.findViewById (R.id.button4) ;
spinnerbtn.setOnClickListener (new OnClickListener(){
     @Override
     public void onClick (View v) {
          // TODO Auto-generated method stub
          Intent intent=new Intent (WidgetDemoActivity.this,SpinnerActivity.class) ;
          startActivity (intent) ;
     }
}) ;
```

同时在 AndroidManifest.xml 文件中声明该 Activity：

```
<activity android:name=" SpinnerActivity "></activity>
```

SpinnerActivity 的运行效果如图 4.21 所示。



图 4.21 Spinner 的应用界面

SpinnerActivity 使用的布局文件为 spiner.xml，其代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match parent"
    android:layout_height="match parent"
    android:orientation="vertical">
     <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap content"
        android:layout_height="wrap content"
        android:text="textview"/>
   <Spinner
        android:id="@+id/spinner1"
        android:layout_width="match_parent"
        android:layout_height="wrap content" />

</LinearLayout>
```

SpinnerActivity.java 文件的代码如下：

```
package introduction.android.widgetDemo;

import java.util.ArrayList;
import java.util.List;
import android.app.Activity;
import android.os.Bundle;
import android.view.MotionEvent;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.Spinner;
import android.widget.TextView;

public class SpinnerActivity extends Activity {
     private List<String>list=new ArrayList<String>();
     private TextView textview;
     private Spinner spinnertext;
     private ArrayAdapter<String>adapter;
     public void onCreate (Bundle savedInstanceState) {
            super.onCreate (savedInstanceState) ;
            setContentView (R.layout.spiner) ;
            //第一步：定义下拉列表内容
            list.add ("沈阳") ;
            list.add ("天津") ;
            list.add ("北京") ;
            list.add ("上海") ;
            list.add ("深圳") ;
            textview= (TextView) findViewById (R.id.textView1) ;
            spinnertext= (Spinner) findViewById (R.id.spinner1) ;
            //第二步：为下拉列表定义一个适配器
            adapter=new ArrayAdapter<String> (this,android.R.layout.simple spinner item,
list) ;
            //第三步：设置下拉列表下拉时的菜单样式
            adapter.setDropDownViewResource
(android.R.layout.simple spinner dropdown item) ;
            //第四步：将适配器添加到下拉列表上
            spinnertext.setAdapter (adapter) ;
```

```
                //第五步：添加监听器，为下拉列表设置事件的响应
                spinnertext.setOnItemSelectedListener (new Spinner.OnItemSelectedListener(){
                    public void onItemSelected (AdapterView<?>arg0, View arg1, int arg2, long arg3)
{
                        // TODO Auto-generated method stub
                        /* 将所选 spinnertext 的值带入 myTextView 中*/
                      textview.setText ("我来自："+adapter.getItem (arg2)) ;
                        /* 将 spinnertext 显示*/
                        arg0.setVisibility (View.VISIBLE) ;
                    }
                    public void onNothingSelected (AdapterView<?>arg0) {
                        // TODO Auto-generated method stub
                      textview.setText ("NONE") ;
                        arg0.setVisibility (View.VISIBLE) ;
                    }
                }) ;
                //将 spinnertext 添加到 OnTouchListener 对内容选项触屏事件处理
                spinnertext.setOnTouchListener (new Spinner.OnTouchListener(){
                    @Override
                    public boolean onTouch (View v, MotionEvent event) {
                        // TODO Auto-generated method stub
                        // 将 mySpinner 隐藏
                          v.setVisibility (View.INVISIBLE) ;
                          Log.i ("spinner","Spinner Touch 事件被触发!") ;
                          return false;
                    }
                }) ;
                //焦点改变事件处理
                spinnertext.setOnFocusChangeListener (new Spinner.OnFocusChangeListener(){
                    public void onFocusChange (View v, boolean hasFocus) {
                        // TODO Auto-generated method stub
                        v.setVisibility (View.VISIBLE) ;
                        Log.i ("spinner","Spinner FocusChange 事件被触发! ") ;
                    }
                }) ;
        }
    }
```

SpinnerActivity 通过 5 个步骤将 Spinner 初始化并进行事件处理，分别为：

- 定义下拉列表的列表项内容 List<String>。
- 为下拉列表 Spinner 定义一个适配器 ArrayAdapter<String>，并与列表项内容相关联。
- 使用 ArrayAdapter.setDropDownViewResource()设置 Spinner 下拉列表在打开时的下拉菜单样式。
- 使用 Spinner. setAdapter()将适配器数据与 Spinner 关联起来。
- 为 Spinner 添加事件监听器，进行事件处理。

Spinner 支持多种事件处理方式，本实例中对 Spinner 被单击事件、焦点改变事件和 Spinner 的列表项被选中事件进行了处理。

在本实例中，SpinnerActivity 在程序代码中动态建立了下拉列表每一项的内容。除此之外，还可以在 XML 文件中定义 Spinner 的下拉列表项，步骤如下。

在 res/values 文件夹下新建 cities.xml 文件夹：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string-array name="city">
     <item>shenyang</item>
     <item>nanjing</item>
     <item>beijing</item>
     <item>tianjin</item>
  </string-array>
</resources>
```

在 SpinnerActivity.java 中初始化 Spinner：

```
Spinner spinner= (Spinner) findViewById (R.id.spinner1) ;
ArrayAdapter<CharSequence>adapter=ArrayAdapter.createF
romResource (this, R.array.city,
android.R.layout.simple_spinner_item) ;
adapter.setDropDownViewResource
(android.R.layout.simple_spinner_dropdown_item) ;
spinner.setAdapter (adapter) ;
```

运行效果如图 4.22 所示。



图 4.22　Spinner 的事件处理

4.4.8　自动完成文本

在使用百度或者 Google 搜索信息时，只需要在搜索框中输入几个关键字，就会有很多相关的信息以列表形式被列举出来供用户选择，这种效果在 Android SDK 中可以通过 AutoCompleteTextView 来实现。

下面用一个简单的实例讲解 AutoCompleteTextView 的使用方法。在工程 WidgetDemo 的布局文件 main.xml 中添加一个 Button，用以启动 AutoCompleteTextViewActivity。

在 main.xml 中添加代码如下：

```
<Button
   android:id="@+id/button5"
   android:layout_width="wrap content"
   android:layout_height="wrap_content"
   android:text="AutoCompleteTextViewDemo" />
```

单击 Button 并启动 AutoCompleteTextViewActivity 的代码如下：

```
Button autobtn= (Button) this.findViewById (R.id.button5) ;
autobtn.setOnClickListener (new OnClickListener(){
     @Override
     public void onClick (View v) {
          // TODO Auto-generated method stub
          Intent intent=new Intent
(WidgetDemoActivity.this,AutoCompleteTextViewActivity.class) ;
          startActivity (intent) ;
     }
}) ;
```

同时在 AndroidManifest.xml 文件中声明该 Activity：

```
<activity android:name=" AutoCompleteTextViewActivity"></activity>
```

AutoCompleteTextViewActivity 的运行效果如图 4.23 所示。

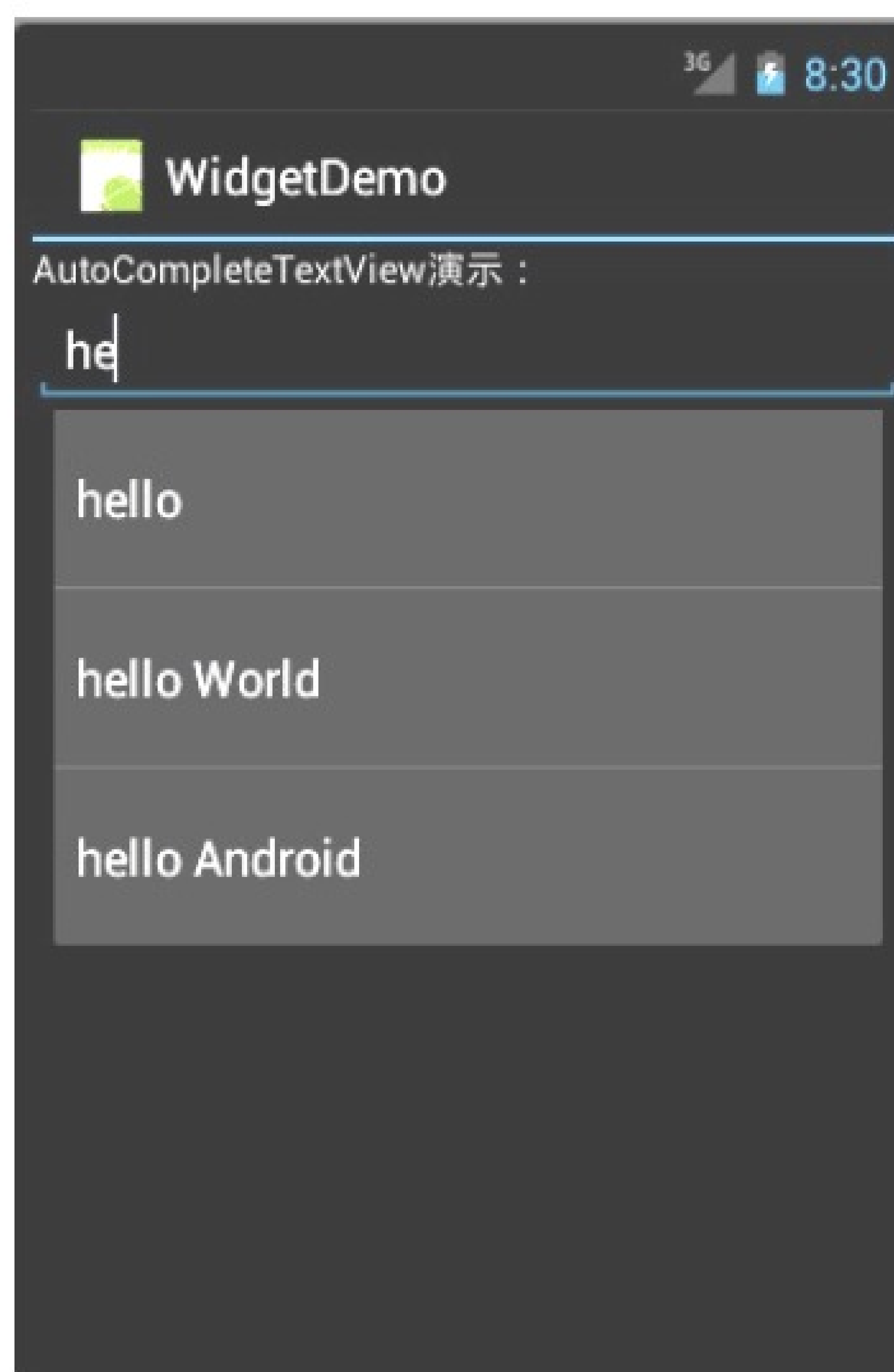


图 4.23 AutoCompleteTextViewActivity 的运行效果

AutoCompleteTextViewActivity 使用的布局文件为 autocompletetextview.xml，其具体内容如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
      <TextView
         android:layout_width="fill_parent"
         android:layout_height="wrap_content"
         android:text="AutoCompleteTextView 演示: " />

   <AutoCompleteTextView
         android:id="@+id/autoCompleteTextView1"
         android:layout_width="match_parent"
         android:layout_height="wrap_content"
         android:text="">
        <requestFocus />
   </AutoCompleteTextView>

</LinearLayout>
```

AutoCompleteTextViewActivity.java 的代码如下：

```
package introduction.android.widgetDemo;

import android.app.Activity;
import android.os.Bundle;
import android.widget.ArrayAdapter;
import android.widget.AutoCompleteTextView;

public class AutoCompleteTextViewActivity extends Activity {
      private AutoCompleteTextView textView;
```

```
            private static final String[] autotext=new String[] {"hello","hello World","hello
Android"};
            @Override
            public void onCreate (Bundle savedInstanceState) {
                super.onCreate (savedInstanceState) ;
                setContentView (R.layout.autocompletetextview) ;
                textView= (AutoCompleteTextView ) findViewById (R.id.autoCompleteTextView1) ;
                /*new ArrayAdapterd 对象将 autotext 字符串数组传入*/
                ArrayAdapter<String>adapter=new ArrayAdapter<String>
(this,android.R.layout.simple_dropdown_item_1line,autotext) ;
                /*将 ArrayAdapter 添加到 AutoCompleteTextView 中*/
                textView.setAdapter (adapter) ;
            }
    }
```

AutoCompleteTextViewActivity 中为可自动补全的内容建立对应字符串数组 autotext，将该数组关联到 ArrayAdapter 中，然后将 ArrayAdapter 与 AutoCompleteTextView 相关联，进而实现自动完成文本功能。

AutoCompleteTextView 提供一系列属性对显示效果进行设置，分别说明如下。

- completionThreshold：它的值决定了你在 AutoCompleteTextView 中至少输入几个字符，才会具有自动提示的功能。另外，默认最多提示 20 条。
- dropDownAnchor：它的值是一个 View 的 ID，指定后，AutoCompleteTextView 会在这个 View 下弹出自动提示。
- dropDownSelector：应该是设置自动提示的背景色之类的，没有尝试过，有待进一步考证。
- dropDownWidth：设置自动提示列表的宽度。

4.4.9　日期选择器和时间选择器

Android SDK 提供了 DatePicker 和 TimePicker 组件，分别对日期和时间进行选择，方便日期和时间设定。

下面用一个简单的实例讲解 DatePicker 和 TimePicker 组件的使用方法。在工程 WidgetDemo 的布局文件 main.xml 中添加一个名为“Date/Time”的 Button，用以启动 TimeActivity。

在 main.xml 中添加代码如下：

```
    <Button
        android:id="@+id/button6"
        android:layout_width="wrap_content"
        android:layout_height="wrap content"
        android:text=" Date/Time " />
```

单击 Button 并启动 TimeActivity 的代码如下：

```
    Button timebtn= (Button) this.findViewById (R.id.button6) ;
          timebtn.setOnClickListener (new OnClickListener() {
                @Override
                public void onClick (View v) {
                     // TODO Auto-generated method stub
                     Intent intent=new Intent (WidgetDemoActivity.this,TimeActivity.class) ;
```

```
                startActivity(intent);
            }
        });
```

同时在 AndroidManifest.xml 文件中声明该 Activity：

```
<activity android:name=" TimeActivity"></activity>
```

TimeActivity 的运行效果如图 4.24 所示。

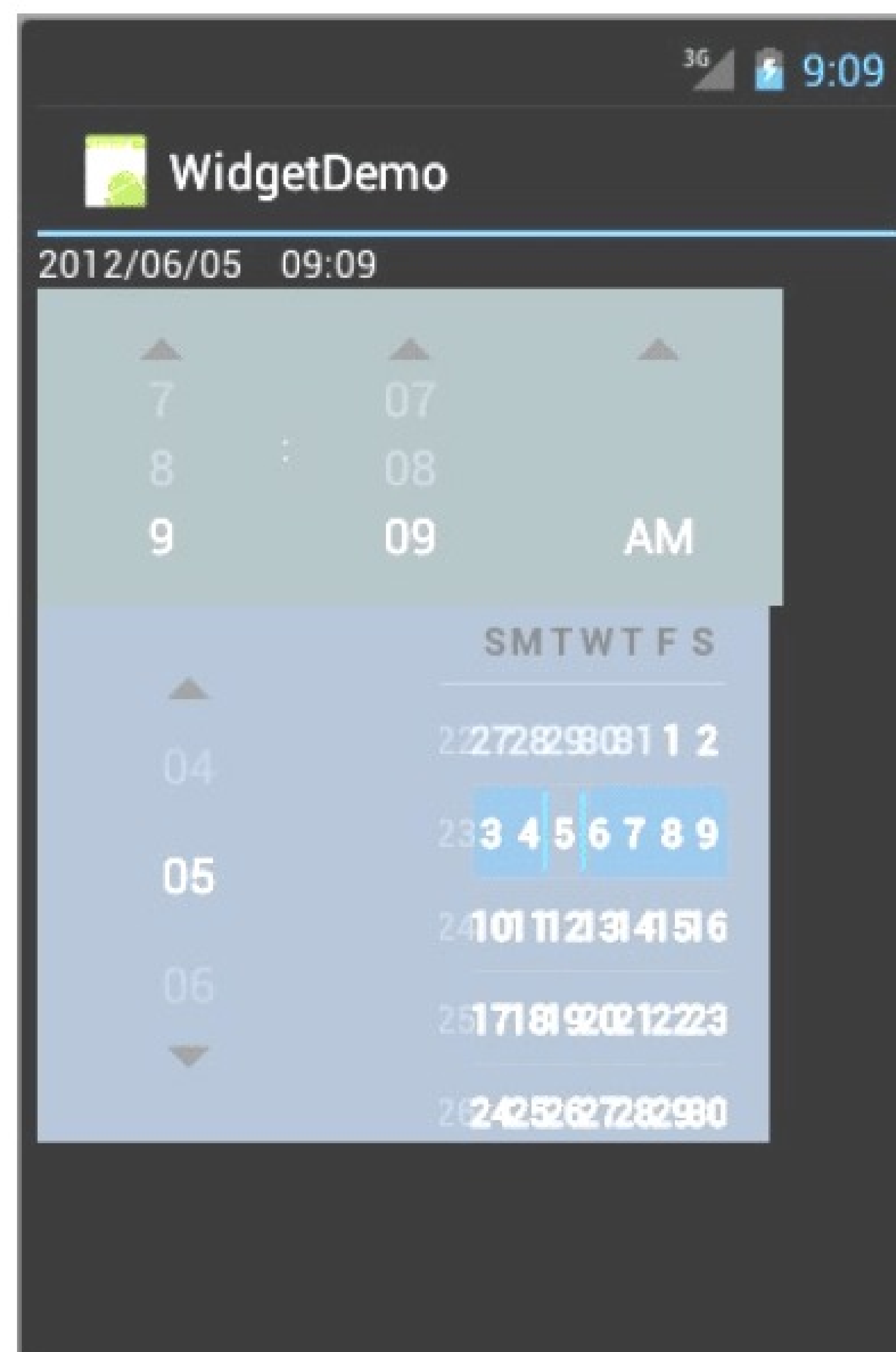


图 4.24 TimeActivity 的运行效果

TimeActivity 使用的布局文件为 time.xml，其内容如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
      <TextView
        android:id="@+id/timeview"
        android:layout_width="fill parent"
        android:layout_height="wrap content"
        android:text="DatePicker 和 TimePicker 演示" />
      <TimePicker
         android:id="@+id/timepicker"
         android:layout_width="wrap_content"
         android:layout_height="116dp"
         android:background="#778888" />
<!-- 设置背景色为墨绿 -->
      <DatePicker
         android:id="@+id/datepicker"
         android:layout_width="271dp"
         android:layout_height="196dp"
         android:background="#778899" />
</LinearLayout>
```

TimeActivity.java 的代码如下：

```
package introduction.android.widgetDemo;

import java.util.Calendar;

import android.app.Activity;
import android.os.Bundle;
import android.widget.DatePicker;
import android.widget.TextView;
import android.widget.TimePicker;

public class TimeActivity extends Activity {
     private TextView textview;
    private TimePicker timepicker;
    private DatePicker datepicker;
    /* 声明日期及时间变量 */
    private int year;
    private int month;
    private int day;
    private int hour;
    private int minute;

    @Override
    public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.time);
      /* 获取当前日期及时间 */
      Calendar calendar=Calendar.getInstance();
      year=calendar.get(Calendar.YEAR);
      month=calendar.get(Calendar.MONTH);
      day=calendar.get(Calendar.DAY_OF_MONTH);
      hour=calendar.get(Calendar.HOUR);
      minute=calendar.get(Calendar.MINUTE);
      datepicker=(DatePicker)findViewById(R.id.datepicker);
      timepicker=(TimePicker)findViewById(R.id.timepicker);
      /* 设置 TextView 对象，显示初始日期时间 */
      textview=(TextView)findViewById(R.id.timeview);
      textview.setText(new StringBuilder().append(year).append("/")
           .append(format(month+1)).append("/").append(format(day))
           .append("  ").append(format(hour)).append(":")
           .append(format(minute)));
      /* 设置 OnDateChangedListener()*/
      datepicker.init(year, month, day,
           new DatePicker.OnDateChangedListener(){
              @Override
              public void onDateChanged(DatePicker view, int year,
                   int monthOfYear, int dayOfMonth) {
               // TODO Auto-generated method stub
               TimeActivity.this.year=year;
               month=monthOfYear;
               day=dayOfMonth;
               textview.setText(new StringBuilder().append(year)
                    .append("/").append(format(month+1))
                    .append("/").append(format(day)).append("  ")
```

```
                        .append (format (hour)) .append (":")
                        .append (format (minute))) ;
                }
            }) ;
        timepicker.setOnTimeChangedListener (new TimePicker.OnTimeChangedListener()
          {

            @Override
            public void onTimeChanged (TimePicker view, int hourOfDay, int minute)
            {
                // TODO Auto-generated method stub
                hour=hourOfDay;
                TimeActivity.this.minute=minute;
                textview.setText (new StringBuilder().append (year)
                 .append ("/") .append (format (month+1))
                 .append ("/") .append (format (day)) .append ("  ")
                 .append (format (hour)) .append (":")
                 .append (format (minute))) ;
            }
          }) ;

    }

    private String format (int time) {
      String str=""+time;
      if (str.length()==1)
          str="0"+str;
      return str;
    }
}
```

TimeActivity 中使用 java.util.Calendar 对象获取当前系统时间。当更改 DatePicker 组件中的日期时，会触发 DatePicker 的 OnDateChange()事件；当修改 TimePacker 的时间时，会触发 TimePacker 的 OnDateChange()事件。

由本实例可见，DatePicker 实现 OnDateChangedListener 监听器的方法与 TimePicker 实现 setOnTimeChangedListener 监听器的方法有所类似。DatePicker 用 init()方法设定年、月、日的同时设定监听器，而 TimePicker 使用 setOnTimeChangedListener()直接设定。

4.4.10 进度条

当应用程序在后台运行时，可以使用进度条（ProgressBar）反馈给用户当前的进度信息。进度条被用以显示当前应用程序的运行状况、功能完成多少等情况。Android SDK 提供两种样式的进度条，一种是圆形的进度条，另一种是水平进度条。其中圆形进度条分大、中、小三种。

进度条本质上是一个整数，显示当前的整数值在特定范围内的比重。下面用一个简单的实例讲解 ProgressBar 组件的使用方法。

在工程 WidgetDemo 的布局文件 main.xml 中添加一个名为 ProgressBarDemo 的 Button，用以启动 ProcessBarActivity。

在 main.xml 中添加代码如下：

```
    <Button
       android:id="@+id/button7"
       android:layout_width="wrap content"
       android:layout_height="wrap content"
       android:text="ProgressBarDemo" />
```

单击 Button 并启动 ProcessBarActivity 的代码如下：

```
    Button processbtn=(Button)this.findViewById(R.id.button7);
          processbtn.setOnClickListener(new OnClickListener(){
                 @Override
                 public void onClick(View v){
                     // TODO Auto-generated method stub
                     Intent intent=new Intent
(WidgetDemoActivity.this,ProcessBarActivity.class);
                     startActivity(intent);
                 }
          });
```

同时在 AndroidManifest.xml 文件中声明该 Activity：

```
    <activity android:name="ProcessBarActivity"></activity>
```

ProcessBarActivity 的运行效果如图 4.25 所示。

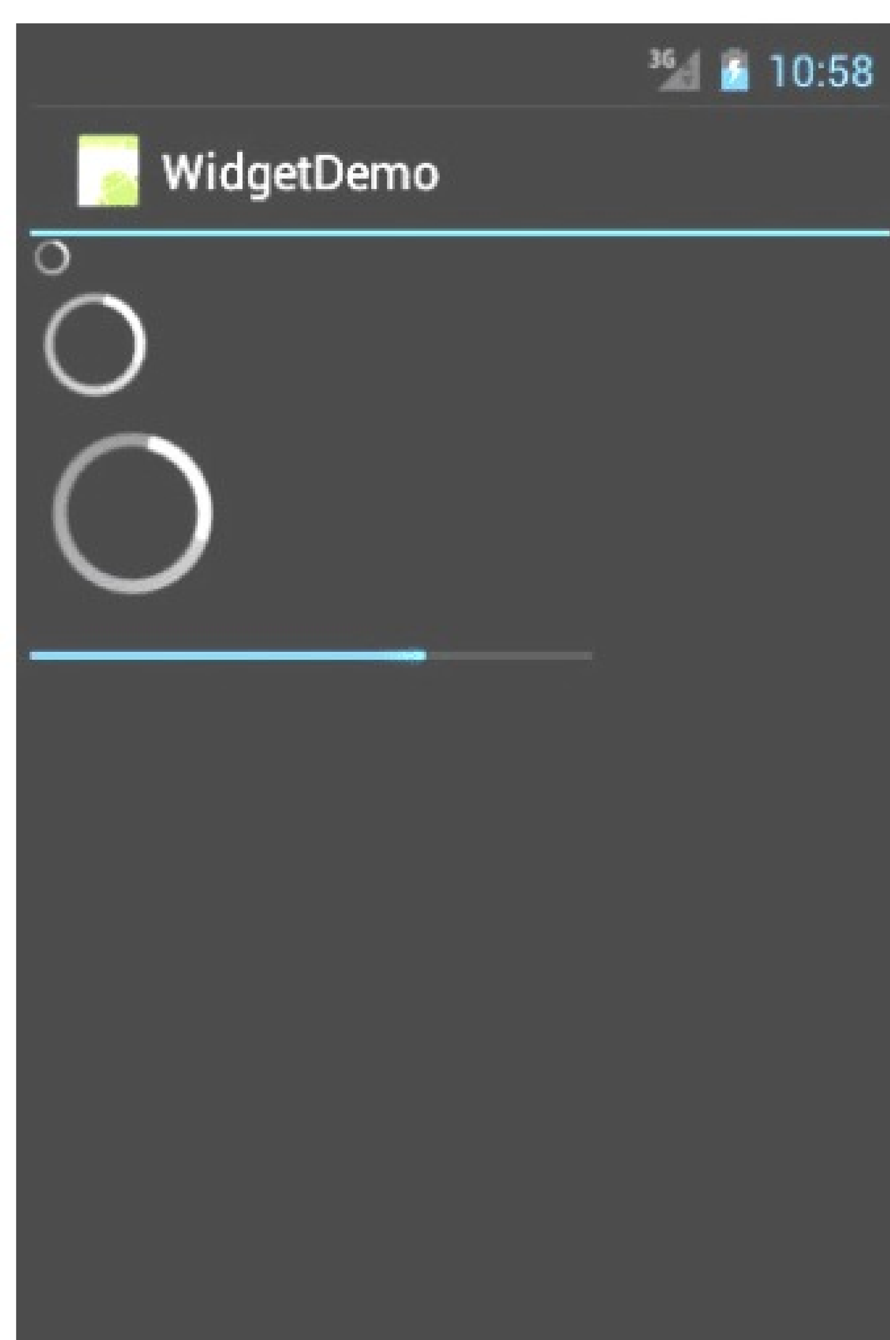


图 4.25　ProcessBarActivity 的运行效果

ProcessBarActivity 使用的布局文件为 processbar.xml，其内容如下：

```
    <?xml version="1.0" encoding="utf-8"?>
    <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
       android:layout_width="match parent"
       android:layout_height="match parent"
       android:orientation="vertical">
        <ProgressBar
          android:id="@+id/progressBar1"
          style="?android:attr/progressBarStyleSmall"
          android:layout_width="wrap content"
          android:layout_height="wrap_content" />
```

```
    <ProgressBar
        android:id="@+id/progressBar2"
        android:layout_width="wrap content"
        android:layout_height="wrap content" />

    <ProgressBar
        android:id="@+id/progressBar3"
        style="?android:attr/progressBarStyleLarge"
        android:layout_width="wrap_content"
        android:layout_height="wrap content" />


    <ProgressBar
        android:id="@+id/progressBar4"
        style="?android:attr/progressBarStyleHorizontal"
        android:layout_width="209dp"
        android:layout_height="30dp"
        android:max="100"/>

</LinearLayout>
```

该布局中放置了小、中、大三种类型的圆形进度条各一个，以及一个水平放置的条形进度条。一般情况下，开发人员不会为圆形进度条指定进度，圆形进度条只是展示运行效果，而不反映实际的进度。条形进度条则不同，开发人员会为条形进度条指定最大值，以及进度条当前值的获取方法。在本实例中，条形进度条的最大值为 100。

ProcessBarActivity.java 的代码如下：

```
package introduction.android.widgetDemo;

import android.app.Activity;
import android.os.Bundle;
import android.os.Handler;
import android.widget.ProgressBar;

public class ProcessBarActivity extends Activity {
    ProgressBar progressBar;
    int i=0;
    int progressBarMax=0;
    /* 创建 Handler 对象 */
    Handler handler=new Handler();

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.processbar);
        progressBar=(ProgressBar) findViewById(R.id.progressBar4);
        /* 获取最大值 */
        progressBarMax=progressBar.getMax();
        /* 匿名内部类启动实现效果的线程 */
        new Thread(new Runnable(){
            @Override
            public void run(){
                while(i++<progressBarMax) {
```

```
                        // 设置滚动条当前状态值
                        progressBar.setProgress(i);
                        try {
                            Thread.sleep(15);

                        } catch(Exception e) {
                            e.printStackTrace();
                        }
                    }
                }
            }).start();
        }
    }
```

ProcessBarActivity 对水平进度条进行了处理。先获取了水平进度条的最大值，然后启动了一个线程，由该线程来控制进度条的值，从 0 开始，每隔 15 毫秒增加 1。

4.4.11　滚动视图

当 Activity 提供的用户界面上有很多内容，以至于当前手机屏幕不能完全显示全部内容时，就需要滚动视图来帮助浏览全部的内容。

以工程 WidgetDemo 为例，由于在讲述过程中不断地在 main.xml 文件中添加按钮和其他组件，目前已经不能显示全部内容，效果如图 4.26 所示。

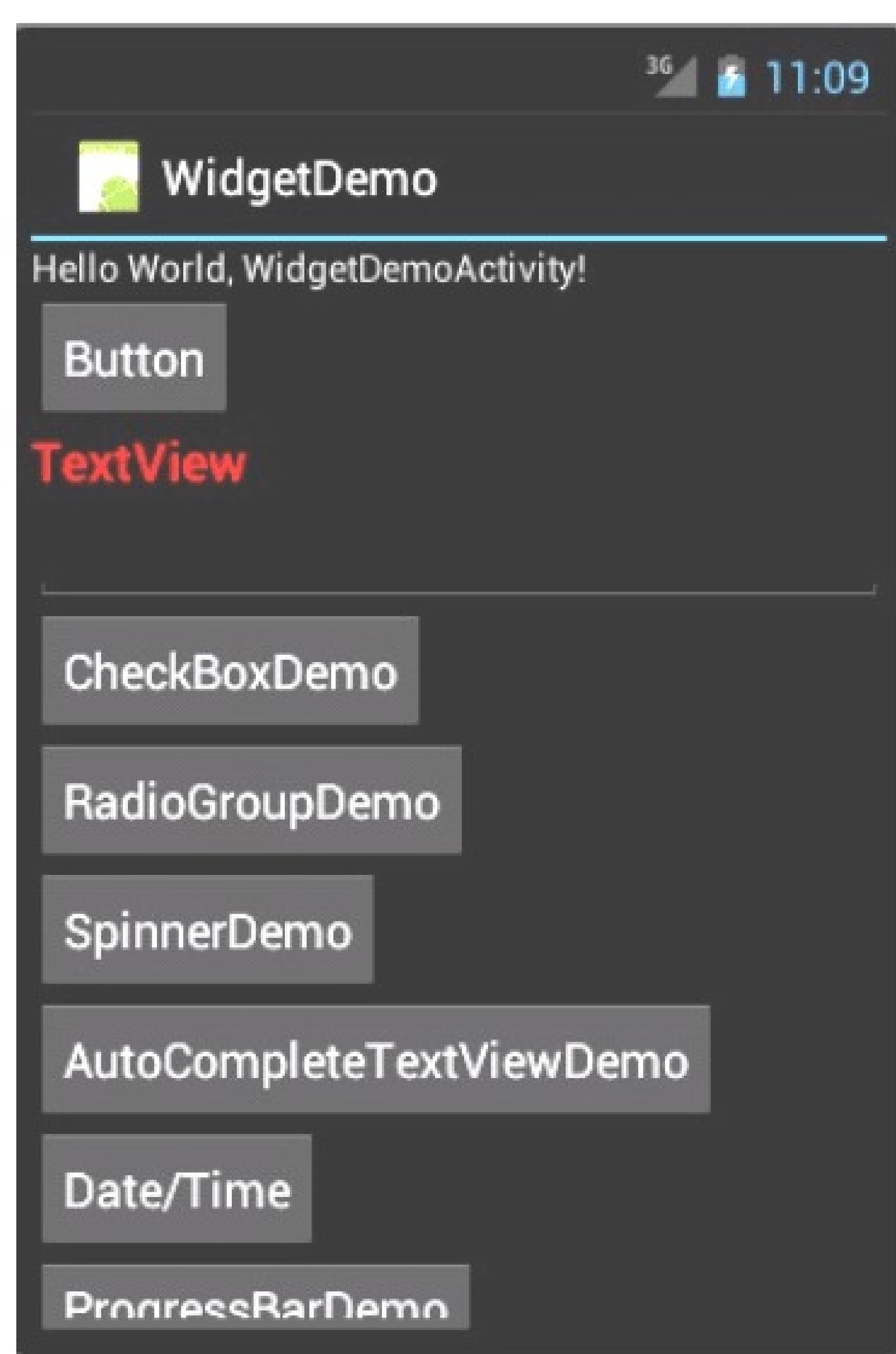


图 4.26　添加大量组件后的效果

这时候就需要使用 ScrollView，即将当前的 Activity 的视图转化为滚动视图，以便于浏览。ScrollView 的使用非常方便，只需在 main.xml 的<LinearLayout>标签外面加上 ScrollView 组件的声明即可。布局文件 main.xml 的内容如下：

```
<ScrollView
    xmlns:android="http://schemas.android.com/apk/res/android"
```

```
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
   <LinearLayout>....</LinearLayout>
</ScrollView>
```

添加 ScrollView 后，main.xml 布局的运行效果如图 4.27 所示。

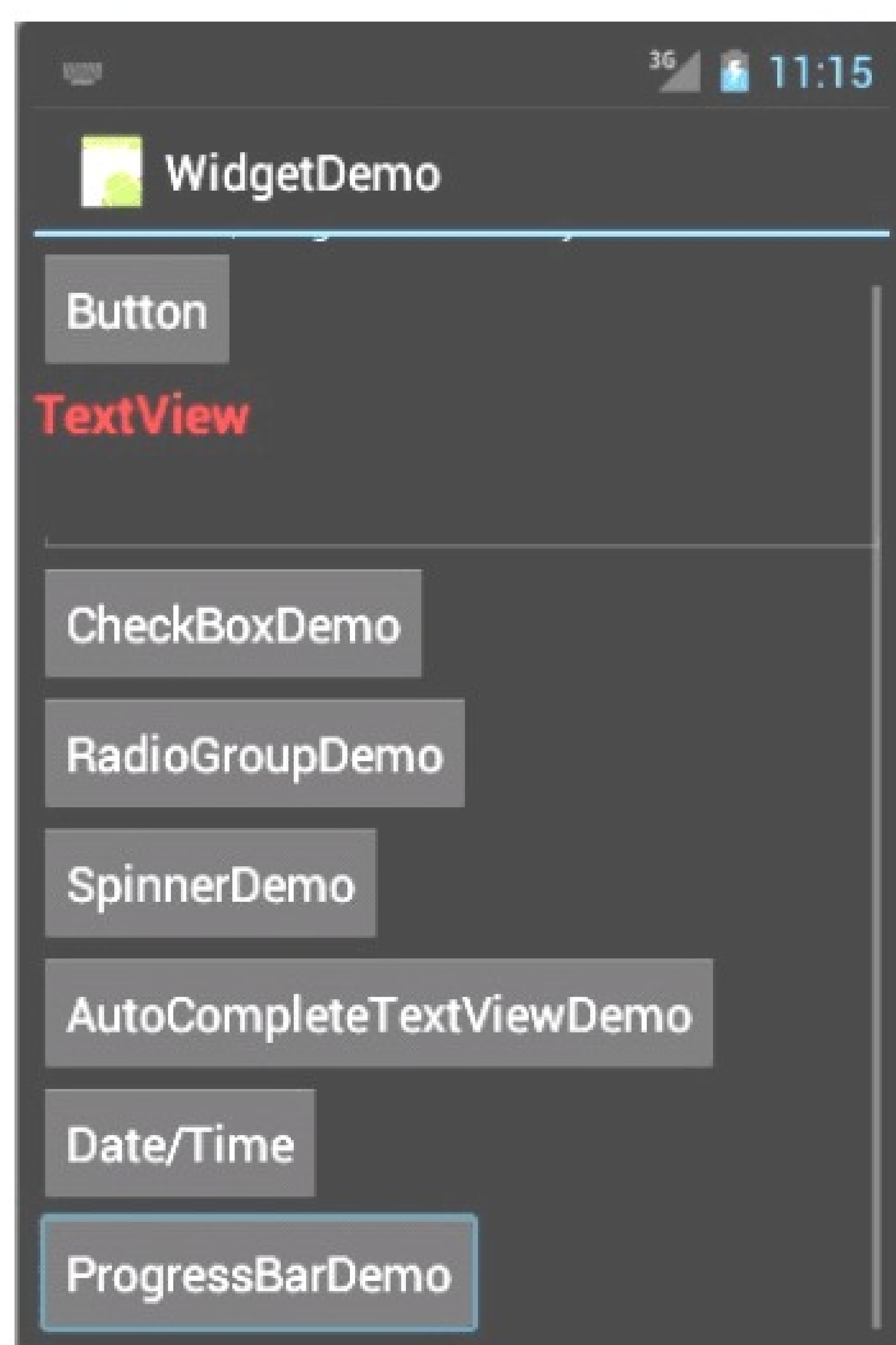


图 4.27　ScrollView 的运行效果

4.4.12　拖动条

SeekBar 是水平进度条 ProgressBar 的间接子类，相当于一个可以拖动的水平进度条。下面仍以一个简单的实例讲解 SeekBar 组件的使用方法。

在工程 WidgetDemo 的布局文件 main.xml 中添加一个名为“SeekBarDemo”的 Button，用以启动 SeekBarActivity。

在 main.xml 中添加代码如下：

```
<Button
    android:id="@+id/button8"
    android:layout_width="wrap content"
    android:layout_height="wrap_content"
    android:text="SeekBarDemo" />
```

单击 Button 并启动 SeekBarActivity 的代码如下：

```
Button processbtn=(Button)this.findViewById(R.id.button8);
processbtn.setOnClickListener(new OnClickListener(){
     @Override
     public void onClick(View v){
          // TODO Auto-generated method stub
          Intent intent=new Intent(WidgetDemoActivity.this, SeekBarActivity.class);
          startActivity(intent);
     }
});
```

同时在 AndroidManifest.xml 文件中声明该 Activity：

```
<activity android:name="SeekBarActivity"></activity>
```

SeekBarActivity 的运行效果如图 4.28 所示。



图 4.28　SeekBarActivity 的运行效果

SeekBarActivity 使用的布局文件为 seekbar.xml，其内容如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

   <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="TextView" />

   <SeekBar
        android:id="@+id/seekBar1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
android:max="100"/>

</LinearLayout>
```

该文件确定 SeekBar 对象的最大值为 100，宽度为手机屏幕的宽度。

SeekBarActivity.java 的代码如下：

```
package introduction.android.widgetDemo;

import android.app.Activity;
```

```
import android.os.Bundle;
import android.util.Log;
import android.widget.SeekBar;
import android.widget.TextView;

public class SeekBarActivity extends Activity {
    private TextView textView;
    private SeekBar seekBar;
    @Override
    public void onCreate (Bundle savedInstanceState) {
        super.onCreate (savedInstanceState) ;
        setContentView (R.layout.seekbar) ;
        textView= (TextView) findViewById (R.id.textView1) ;
        seekBar= (SeekBar) findViewById (R.id.seekBar1) ;
        /* 设置 SeekBar 监听 setOnSeekBarChangeListener */
        seekBar.setOnSeekBarChangeListener (new SeekBar.OnSeekBarChangeListener(){
            /* 拖动条停止拖动时调用 */
            @Override
            public void onStopTrackingTouch (SeekBar seekBar) {
                Log.i ("SeekBarActivity", "拖动停止") ;
            }
            /* 拖动条开始拖动时调用 */
            @Override
            public void onStartTrackingTouch (SeekBar seekBar) {
                Log.i ("SeekBarActivity", "开始拖动") ;
            }
            /* 拖动条进度改变时调用 */
            @Override
            public void onProgressChanged (SeekBar seekBar, int progress,
                boolean fromUser) {
                textView.setText ("当前进度为: "+progress+"%") ;
            }
        }) ;
    }
}
```

SeekBar 的事件处理接口为 OnSeekBarChangeListener，该监听器提供对三种事件的监听，分别为当 SeekBar 的拖动条开始被拖动时、拖动条拖动停止时和拖动条的位置发生改变时。SeekBarActivity 在拖动条开始被拖动和拖动停止时，会通过 Logcat 打印相关信息。当拖动条位置发生改变时，将当前的数值显示到 TextView 中。

4.4.13 评价条

在网上购物的时候，经常会对所购买的商品进行打分。一般对商品的评价和打分是以 5 个星星的方式进行的。Android SDK 提供了 RatingBar 组件来实现该功能。

RatingBar 是 SeekBar 和 ProgressBar 的扩展，是 ProgressBar 的间接子类，可以使用 ProgressBar 相关的属性。RatingBar 有三种风格，分别为默认风格（ratingBarStyle）、小风格（ratingBarStyleSmall）和大风格（ratingBarStyleIndicator ）。其中，默认风格的 RatingBar 是我们通常使用的，可以进行交互，而其他两种不能进行交互。

以一个简单的实例讲解 RatingBar 组件的使用方法。在工程 WidgetDemo 的布局文件 main.xml 中添加一个名为“RatingBarDemo”的 Button，用以启动 RatingBarActivity。

在 main.xml 中添加代码如下：

```
<Button
    android:id="@+id/button9"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="RatingBarDemo" />
```

单击 Button 并启动 RatingBarActivity 的代码如下：

```
Button ratingbarbtn=(Button)this.findViewById(R.id.button9);
ratingbarbtn.setOnClickListener(new OnClickListener(){
    @Override
    public void onClick(View v){
        // TODO Auto-generated method stub
        Intent intent=new Intent(WidgetDemoActivity.this,RatingBarActivity.class);
        startActivity(intent);
    }
});
```

同时在 AndroidManifest.xml 文件中声明该 Activity：

```
<activity android:name="RatingBarActivity"></activity>
```

RatingBarActivity 的运行效果如图 4.29 所示。



图 4.29 RatingBarActivity 的运行效果

RatingBarActivity 使用的布局文件 ratingbar.xml 的内容如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="vertical">
```

```
        <TextView
            android:id="@+id/textView1"
            android:layout_width="wrap content"
            android:layout_height="wrap content"
            android:text="TextView" />

        <RatingBar
            android:id="@+id/ratingBar1"
            android:layout_width="wrap_content"
            android:layout_height="wrap content"
            android:numStars="5"
            android:stepSize="0.5"
            android:rating="3"/>

    </LinearLayout>
```

该布局文件使用 LinearLayout 布局，其中放置了一个 TextView 和一个 RatingBar，并对 RatingBar 的相关属性进行了设置。android:numStars="5"用于设置 RatingBar 显示的星星数量为 5 个；android:stepSize="0.5"用于设置 RatingBar 的最小变化单位为半个星星；android:rating ="3"表示 RatingBar 在初始状态下被选中的星星数量为 3 个。

RatingBarActivity.java 的代码如下：

```
    package introduction.android.widgetDemo;

    import android.app.Activity;
    import android.os.Bundle;
    import android.util.Log;
    import android.view.MotionEvent;
    import android.view.View;
    import android.view.View.OnTouchListener;
    import android.widget.RatingBar;
    import android.widget.RatingBar.OnRatingBarChangeListener;
    import android.widget.TextView;
    import android.widget.Toast;

    public class RatingBarActivity extends Activity {
         private RatingBar chooseRatingBar;
         private TextView textView;
         @Override
         public void onCreate (Bundle savedInstanceState) {
              super.onCreate (savedInstanceState) ;
              setContentView (R.layout.ratingbar) ;
              textView= (TextView) findViewById (R.id.textView1) ;
              chooseRatingBar= (RatingBar) findViewById (R.id.ratingBar1) ;

               /*创建 RatingBar 监听器 */
              chooseRatingBar.setOnRatingBarChangeListener (new
    OnRatingBarChangeListener(){

                    @Override
                   public void onRatingChanged (RatingBar ratingBar,float rating, boolean fromUser)
{
                        chooseRatingBar= (RatingBar) findViewById (R.id.ratingBar1) ;
```

```
                    chooseRatingBar.setRating (rating) ;
                    textView.setText ("您选择了"+rating+"个星星") ;
                }
            }) ;
        }
    }
```

RatingBarActivity 为 RatingBar 对象设置了 OnRatingBarChangeListener 监听器，当用户单击 RatingBar 引起被选中星星数量的变化时，该接口会监测到该事件，并且调用 onRatingChanged()方法，更新 TextView 显示的内容。

onRatingChanged()的三个参数所对应的含义如下。

- ratingBar：多个 RatingBar 可以同时指定同一个 RatingBar 监听器。该参数就是当前触发 RatingBar 监听器的那个 RatingBar 对象。
- rating：当前评级分数。取值范围从 0 到 RatingBar 的总星星数。
- fromUser：如果触发监听器的是用户触屏单击或轨迹球左右移动，则为 true。

4.4.14　图片视图和图片按钮

ImageView 是用于显示图片的组件，在很多场合都有比较普遍的使用。ImageView 可以显示任意图像，加载各种来源的图片（如资源或图片库）。ImageView 可以负责计算图片的尺寸，以便在任意的布局中使用，并且可以提供缩放或者着色等选项供开发者使用。

ImageButton 是 ImageView 的子类，相当于一个表明是图片而不是文字的 Button。其使用方法和 Button 完全相同。

下面通过一个实例来了解一下这两个组件的使用方法。在工程 WidgetDemo 的布局文件 main.xml 中添加一个名为 ImageButtonDemo 的 Button，用以启动 ImageButtonActivity。

在 main.xml 中添加代码如下：

```
<Button
       android:id="@+id/button10"
       android:layout_width="wrap_content"
       android:layout_height="wrap content"
       android:text="ImageButtonDemo" />
```

单击 Button 并启动 RatingBarActivity 的代码如下：

```
Button imgbtn= (Button) this.findViewById (R.id.button10) ;
imgbtn.setOnClickListener (new OnClickListener(){
     @Override
     public void onClick (View v) {
          // TODO Auto-generated method stub
          Intent intent=new Intent (WidgetDemoActivity.this,ImageButtonActivity.class) ;
          startActivity (intent) ;
     }
}) ;
```

同时在 AndroidManifest.xml 文件中声明该 Activity：

```
<activity android:name="ImageButtonActivity"></activity>
```

ImageButtonActivity 的运行效果如图 4.30 所示。

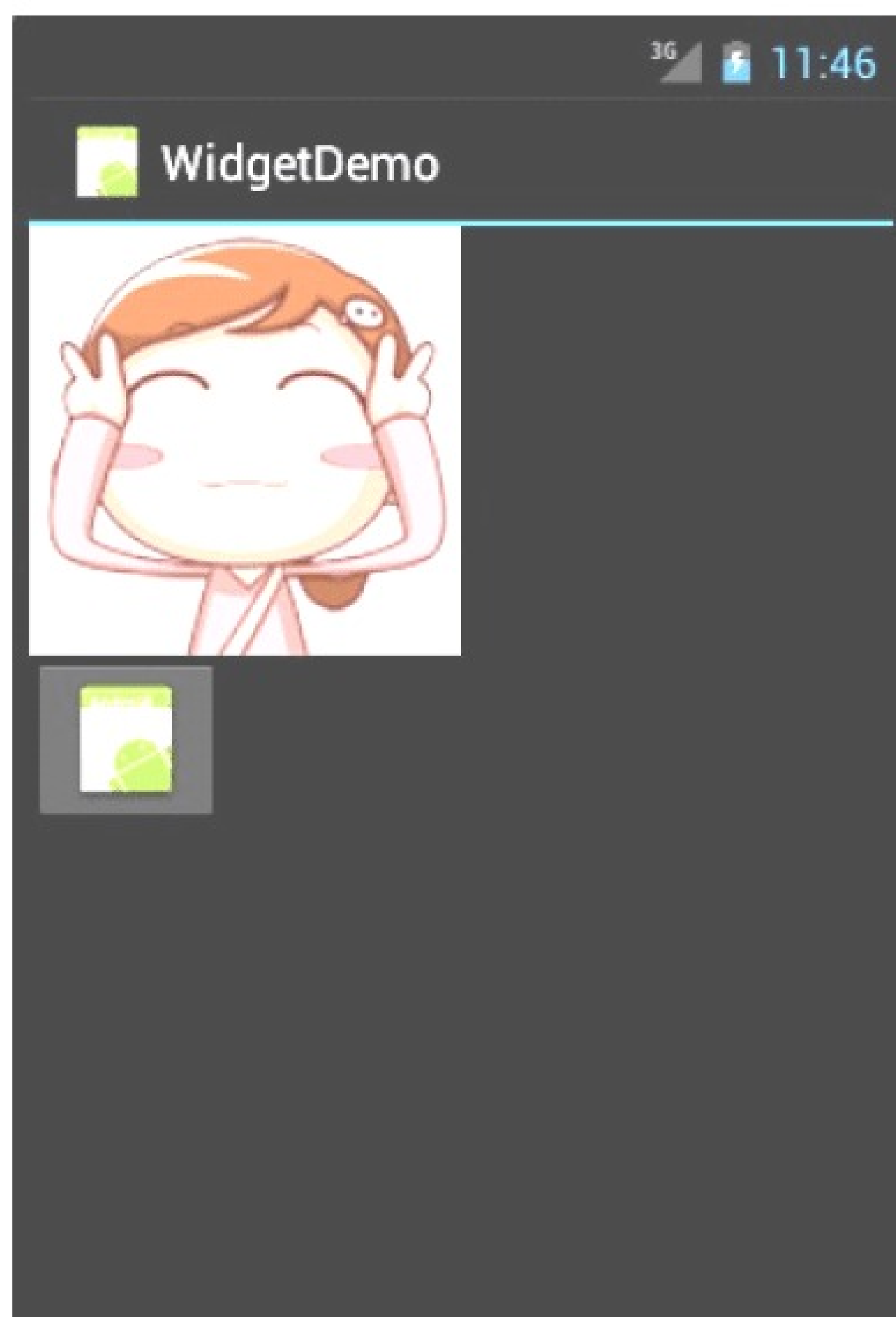


图 4.30　ImageButtonActivity 的运行效果

ImageButtonActivity 的布局文件 imgbtn.xml 内容如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

   <ImageView
        android:id="@+id/imageView1"
        android:layout_width="250dp"
        android:layout_height="250dp"
        android:src="@drawable/girl" />

   <ImageButton
        android:id="@+id/imageButton1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/ic_launcher" />

</LinearLayout>
```

该文件使用 LinearLayout 布局，其中放入了一个 ImageView 组件和一个 ImageButton 组件。两个组件都通过 android:src 属性指定了显示的图片。该实例用到了两个图片资源，一个为 girl，另一个为 ic_launcher，如图 4.31 所示。由于 Android 会根据手机设备的配置高低选择不同的资源，因此为了应用程序的通用性，在三个 drawable 文件夹下都放置了 girl.gif 图像。ic_launcher.png 是系统自带的资源文件。

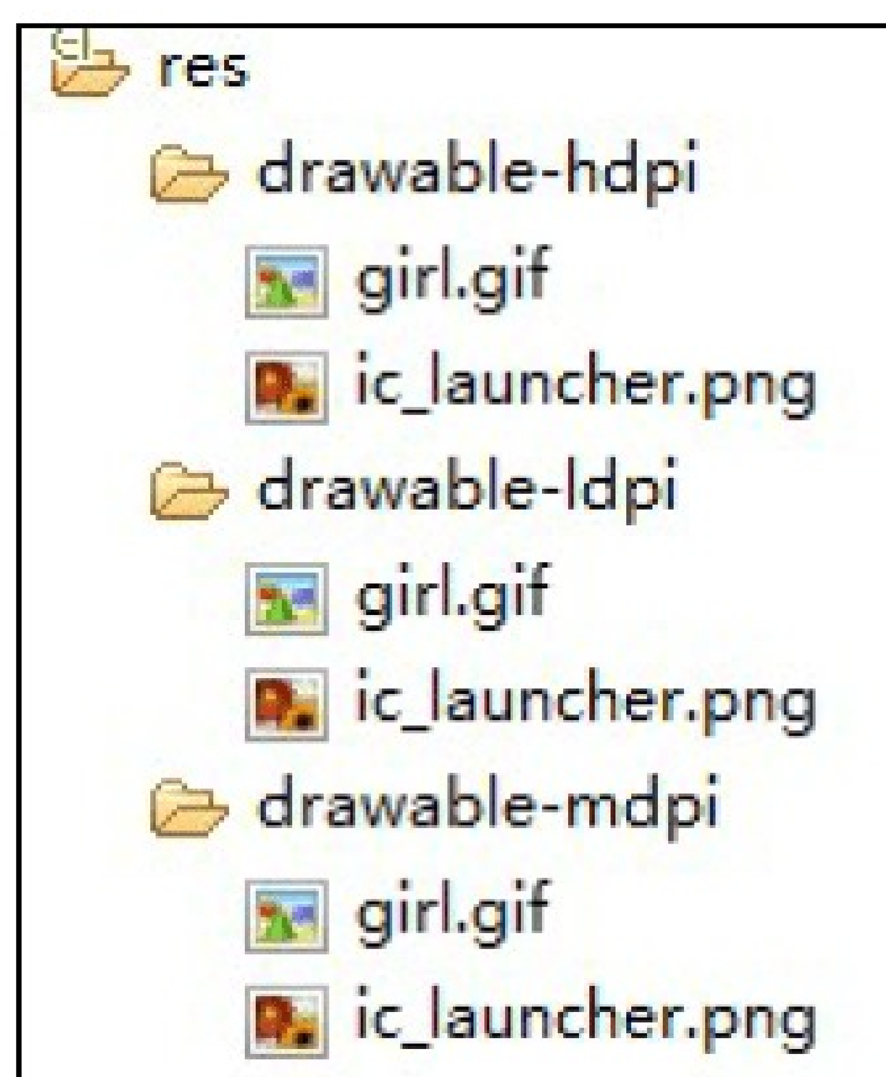


图 4.31　工程中的图片资源

ImageButtonActivity.java 的代码如下：

```
package introduction.android.widgetDemo;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.ViewGroup.LayoutParams;
import android.widget.ImageButton;
import android.widget.ImageView;
public class ImageButtonActivity extends Activity {
     private ImageButton imgbtn;
     private ImageView imgview;
     @Override
     protected void onCreate (Bundle savedInstanceState) {
          // TODO Auto-generated method stub
          super.onCreate (savedInstanceState) ;
          setContentView (R.layout.imgbtn) ;
          imgbtn= (ImageButton) this.findViewById (R.id.imageButton1) ;
          imgview= (ImageView) this.findViewById (R.id.imageView1) ;
          imgbtn.setOnClickListener (new View.OnClickListener(){
               @Override
               public void onClick (View v) {
                    // TODO Auto-generated method stub
                    LayoutParams params=imgview.getLayoutParams();
                    params.height+=3;
                    params.width+=3;
                    imgview.setLayoutParams (params) ;
               }
          }) ;
     }
}
```

ImageButtonActivity 为 ImageButton 添加了单击监听器，对用户单击 imgbtn 的事件进行了处理。用户每次单击图片按钮，都把 ImageView 组件的宽和高增大 3。随着用户的不断单击，ImageView 中显示的图片越来越大，显示了 ImageView 组件对图片的缩放功能。

4.4.15 图片切换器和图库

在使用 Android 手机设置壁纸的时候，会看到屏幕底部有很多可以滚动的图片，当单击某一图片时，在其上面的空间会显示当前选中的图片，此时我们用到的就是 Gallery（图库）和 ImageSwitcher（图片切换器）。

Gallery 组件用于横向显示图像列表，并且自动将当前图像放置到中间位置。ImageSwitcher 则像是图片浏览器，可以切换图片，通过它可以制作简单的幻灯片等。通常将这两个类结合在一起使用，可以制作有一定效果的相册。

下面通过一个实例来了解一下这两个组件的使用方法。

在工程 WidgetDemo 的布局文件 main.xml 中添加一个名为 GalleryDemo 的 Button，用以启动 GalleryActivity。在 main.xml 中添加代码如下：

```
<Button
   android:id="@+id/button11"
   android:layout_width="wrap_content"
   android:layout_height="wrap content"
   android:text="GalleryDemo" />
```

单击 Button 并启动 GalleryActivity 的代码如下：

```
Button gallerybtn=(Button)this.findViewById
(R.id.button11);
    gallerybtn.setOnClickListener(new
OnClickListener(){
        @Override
        public void onClick(View v){
            // TODO Auto-generated method stub
            Intent intent=new Intent
(WidgetDemoActivity.this,GalleryActivity.class);
                startActivity(intent);
            }
    });
```

同时在 AndroidManifest.xml 文件中声明该 Activity：

```
<activity
android:name="GalleryActivity"></activity>
```

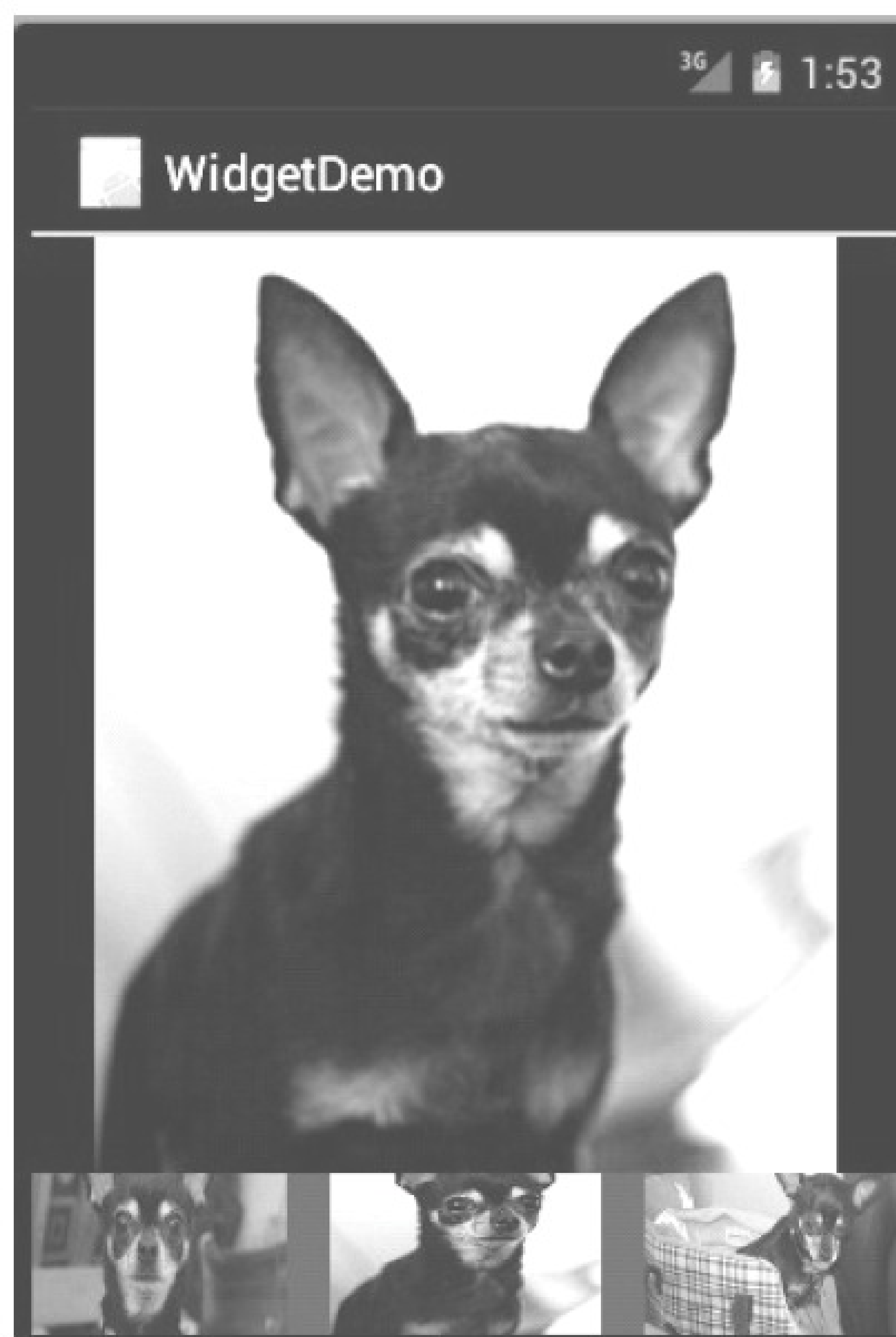


图 4.32 GalleryActivity 的运行效果

GalleryActivity 的运行效果如图 4.32 所示。

GalleryActivity 使用的布局文件为 gallery.xml，内容如下：

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:orientation="vertical">
  <ImageSwitcher
      android:id="@+id/switcher"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
```

```
        android:layout_alignParentTop="true"
        android:layout_alignParentLeft="true">
    </ImageSwitcher>
     <Gallery
     android:id="@+id/gallery"
     android:background="#333333"
     android:layout_width="fill_parent"
     android:layout_height="60dp"
     android:layout_alignParentBottom="true"
     android:layout_alignParentLeft="true"
     android:gravity="center_vertical"
     android:spacing="16dp" />
</RelativeLayout>
```

该布局文件使用的是相对布局，通过 android:layout_alignParentTop="true"属性将 ImageSwitcher 放置于视图的顶端，其顶部与其父组件的顶部对齐，同时使用 android: layout_alignParentLeft="true"属性使 ImageSwitcher 的左边缘与其父组件的左边缘对齐。在设置 Gallery 组件时，将其与屏幕的左下角对其，android:layout_alignParentBottom="true"是将该组件的底部与其父组件的底部对齐，并且使用 android:spacing="16dp"属性设置图片之间的间距。

GalleryActivity.java 的代码如下：

```java
package introduction.android.widgetDemo;

import android.app.Activity;
import android.content.Context;
import android.os.Bundle;
import android.view.View;
import android.view.ViewGroup;
import android.view.ViewGroup.LayoutParams;
import android.view.animation.AnimationUtils;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemSelectedListener;
import android.widget.BaseAdapter;
import android.widget.Gallery;
import android.widget.ImageSwitcher;
import android.widget.ImageView;
import android.widget.ViewSwitcher.ViewFactory;

public class GalleryActivity extends Activity {
     private Gallery gallery;
     private ImageSwitcher imageSwitcher;
     private int[] resids=new int[] {
              R.drawable.sample_0, R.drawable.sample_1,
          R.drawable.sample_2, R.drawable.sample_3,
          R.drawable.sample_4, R.drawable.sample_5,
          R.drawable.sample_6, R.drawable.sample_7};
     @Override
     public void onCreate(Bundle savedInstanceState) {
          super.onCreate(savedInstanceState);
          setContentView(R.layout.gallery);
          /* 加载 Gallery 和 ImageSwitcher */
          gallery=(Gallery) findViewById(R.id.gallery);
          imageSwitcher=(ImageSwitcher) findViewById(R.id.switcher);
```

```
        /* 创建用于描述图像数据的 ImageAdapter 对象 */
        ImageAdapter imageAdapter=new ImageAdapter(this);
        /* 设置 Gallery 组件的 Adapter 对象 */
        gallery.setAdapter(imageAdapter);
        /* 添加 Gallery 监听器 */
        gallery.setOnItemSelectedListener(new OnItemSelectedListener(){

            @Override
            public void onItemSelected(AdapterView<?>parent, View view,
                    int position, long id){
                // TODO Auto-generated method stub
                // 当选取 Grallery 上的图片时，在 ImageSwitcher 组件中显示该图像
                imageSwitcher.setImageResource(resids[position]);
            }

            @Override
            public void onNothingSelected(AdapterView<?>arg0){
                // TODO Auto-generated method stub
            }

        });
        /* 设置 ImageSwitcher 组件的工厂对象 */
        imageSwitcher.setFactory(new ViewFactory(){
            /* ImageSwitcher 用这个方法来创建一个 View 对象去显示图片 */
            @Override
            public View makeView(){
                // TODO Auto-generated method stub
                ImageView imageView=new ImageView(GalleryActivity.this);
                /* setScaleType 可以设置当图片大小和容器大小不匹配时的剪辑模式 */
                imageView.setScaleType(ImageView.ScaleType.FIT_CENTER);
                imageView.setLayoutParams(new ImageSwitcher.LayoutParams(
                        LayoutParams.FILL_PARENT, LayoutParams.FILL_PARENT));
                return imageView;
            }

        });
        /* 设置 ImageSwitcher 组件显示图像的动画效果 */
        imageSwitcher.setInAnimation(AnimationUtils.loadAnimation(this,
                android.R.anim.fade_in));
        imageSwitcher.setOutAnimation(AnimationUtils.loadAnimation(this,
                android.R.anim.fade_out));
    }

    public class ImageAdapter extends BaseAdapter {
        /* 定义 Context */
        private Context mContext;

        /* 声明 ImageAdapter */
        public ImageAdapter(Context context){
            mContext=context;
        }

        @Override
        /* 获取图片的个数 */
        public int getCount(){
```

```
            // TODO Auto-generated method stub
            return resids.length;
        }

        /* 获取图片在库中的位置 */
        @Override
        public Object getItem (int position) {
            // TODO Auto-generated method stub
            return position;
        }

        /* 获取图片 ID */
        @Override
        public long getItemId (int position) {
            // TODO Auto-generated method stub
            return position;
        }

        /* 返回具体位置的 ImageView 对象 */
        @Override
        public View getView (int position, View convertView, ViewGroup parent) {
            ImageView imageview=new ImageView (mContext) ;
            /* 给 ImageView 设置资源 */
            imageview.setImageResource (resids[position]) ;
            /* 设置图片布局大小为 100*100 */
            imageview.setLayoutParams (new Gallery.LayoutParams (100, 100)) ;
            /* 设置显示比例类型 */
            imageview.setScaleType (ImageView.ScaleType.FIT_XY) ;
            return imageview;
        }
    }
}
```

Gallery 要显示的图片来自资源文件。把需要显示的图片放在/res/drawable 目录下后，将这些图片的 ID 保存在一个 int 数组中以备使用。相关代码如下：

```
private int[] resids=new int[] {
    R.drawable.sample_0, R.drawable.sample_1,
    R.drawable.sample_2, R.drawable.sample_3,
    R.drawable.sample_4, R.drawable.sample_5,
    R.drawable.sample_6, R.drawable.sample_7};
```

Gallery 通过 setAdapter（imageAdapter）方法将组件和要显示的图片关联起来。本实例中为 Gallery 设定的适配器为 ImageAdapter，主要用于描述图像信息，其为 android.widget.BaseAdapter 的子类。

在 ImageAdapter 类中有两个方法值得我们注意，其中一个是 getCount()方法，它用于返回图片的总数，通常使用获取存放图片数组长度的方法获取图片总数，也可以规定具体的返回数，但不能超过实际图片数量；getView()方法是当 Gallery 中需要显示某一个图像时，将当前图片的索引，也就是 position 的值传入，从 resids 数组中获得相应的图片的 ID。

GalleryActivity 为添加 Gallery 监听器，处理了用户单击 Gallery 中图片的事件，并设置了 ImageSwitcher 相关属性。其代码如下：

```
imageSwitcher.setInAnimation(AnimationUtils.loadAnimation(this,
                    android.R.anim.fade_in));

imageSwitcher.setOutAnimation(AnimationUtils.loadAnimation(this,
                    android.R.anim.fade_out));
```

设置了 ImageSwitcher 组件图片切换时的渐入和渐出效果。

4.4.16 网格视图

GridView 提供了一个二维的可滚动的网格，按照行列的方式来显示内容，一般适合显示图标、图片等，适合浏览。

下面通过一个实例来了解一下 GridView 组件的使用方法。在工程 WidgetDemo 的布局文件 main.xml 中添加一个名为 GridViewDemo 的 Button，用以启动 GridViewActivity。

在 main.xml 中添加代码如下：

```
<Button
    android:id="@+id/button12"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="GridViewDemo" />
```

单击 Button 并启动 GridViewActivity 的代码如下：

```
    Button gridviewbtn=(Button)this.findViewById
(R.id.button12);
    gridviewbtn.setOnClickListener(new
OnClickListener(){
        @Override
        public void onClick(View v){
            // TODO Auto-generated method stub
            Intent intent=new Intent
(WidgetDemoActivity.this,GridViewActivity.class);
                startActivity(intent);
        }
    });
```

同时在 AndroidManifest.xml 文件中声明该 Activity：

```
    <activity android:name="GridViewActivity">
</activity>
```

GridViewActivity 的运行效果如图 4.33 所示。

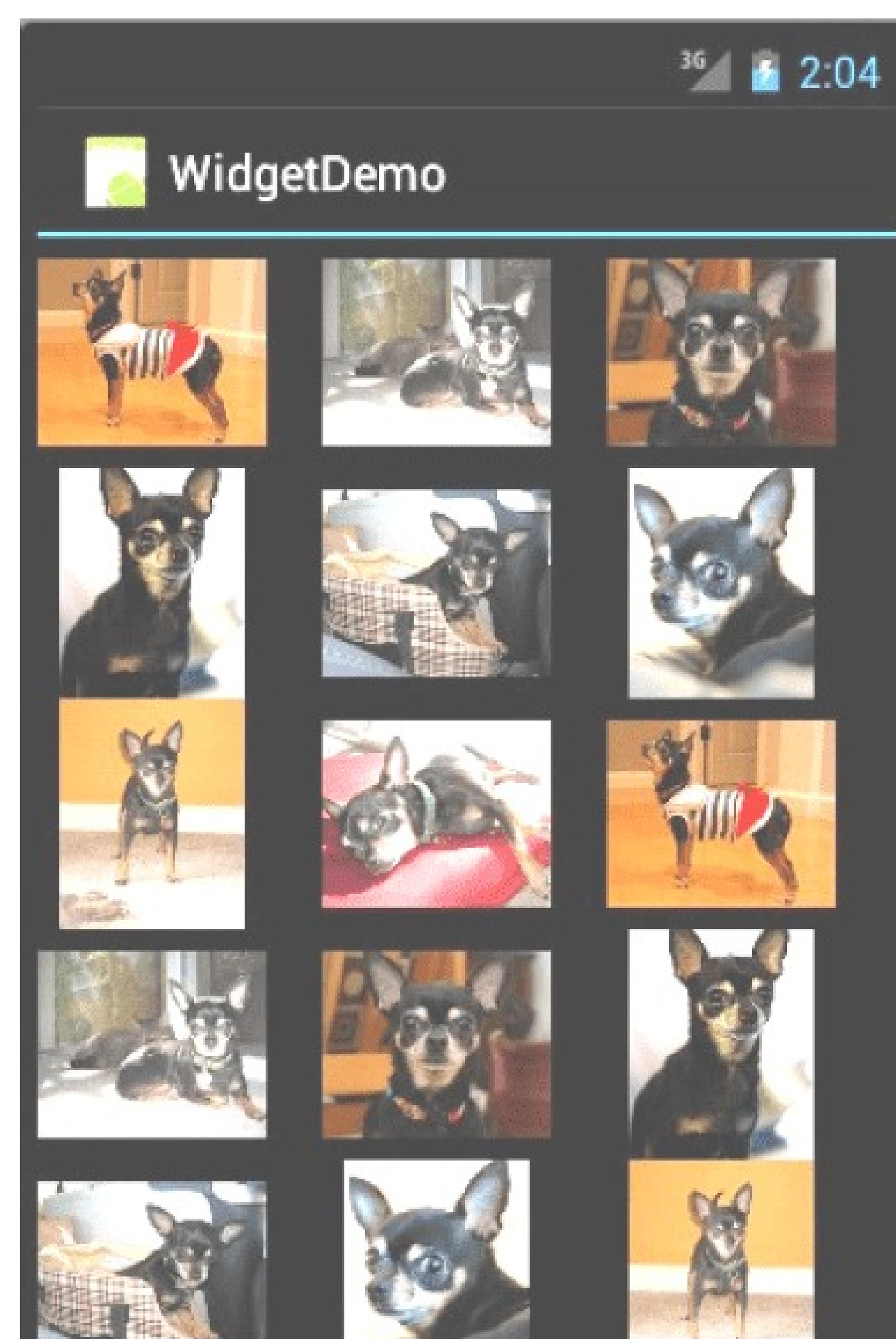


图 4.33 GridViewActivity 的运行效果

GridViewActivity 使用的布局文件为 gridview.xml，其内容如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

   <GridView
```

```
        android:id="@+id/gridView1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:numColumns="3">
    </GridView>

</LinearLayout>
```

该视图采用 LinearLayout 的布局方式，其中放置了一个 GridView 组件，该组件由三列组成。GridViewActivity.java 的代码如下：

```
package introduction.android.widgetDemo;

import android.app.Activity;
import android.content.Context;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.view.ViewGroup;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.BaseAdapter;
import android.widget.GridView;
import android.widget.ImageView;

public class GridViewActivity extends Activity {
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.gridview);

        GridView gridview=(GridView) findViewById(R.id.gridView1);
        gridview.setAdapter(new ImageAdapter(this));

        gridview.setOnItemClickListener(new OnItemClickListener(){
            public void onItemClick(AdapterView<?>parent, View v,
                    int position, long id) {
                Log.i("gridview", "这是第"+position+"幅图像。");
            }
        });
    }

    public class ImageAdapter extends BaseAdapter {
        private Context mContext;

        public ImageAdapter(Context c) {
            mContext=c;
        }

        /* 获取当前图片数量 */
        @Override
        public int getCount(){
            return mThumbIds.length;
        }

        /* 根据需要 position 获得在 GridView 中的对象 */
```

```
            @Override
            public Object getItem(int position) {
                return position;
            }

            /* 获得在 GridView 中对象的 ID */
            @Override
            public long getItemId(int id) {
                return id;
            }

            @Override
            public View getView(int position, View convertView, ViewGroup parent) {
                ImageView imageView;
                if (convertView==null) {
                    /* 实例化 ImageView 对象 */
                    imageView=new ImageView(mContext);
                    /* 设置 ImageView 对象布局，设置 View 的 height 和 width */
                    imageView.setLayoutParams(new GridView.LayoutParams(85, 85));
                    /* 设置边界对齐 */
                    imageView.setAdjustViewBounds(false);
                    /* 按比例统一缩放图片（保持图片的尺寸比例）*/
                    imageView.setScaleType(ImageView.ScaleType.CENTER_CROP);
                    /* 设置间距 */
                    imageView.setPadding(8, 8, 8, 8);
                } else {
                    imageView=(ImageView) convertView;
                }
                imageView.setImageResource(mThumbIds[position]);
                return imageView;
            }
        }

        // references to our images
        private Integer[] mThumbIds={ R.drawable.sample_2, R.drawable.sample_3,
                R.drawable.sample_4, R.drawable.sample_5, R.drawable.sample_6,
                R.drawable.sample_7, R.drawable.sample_0, R.drawable.sample_1,
                R.drawable.sample_2, R.drawable.sample_3, R.drawable.sample_4,
                R.drawable.sample_5, R.drawable.sample_6, R.drawable.sample_7,
                R.drawable.sample_0, R.drawable.sample_1, R.drawable.sample_2,
                R.drawable.sample_3, R.drawable.sample_4, R.drawable.sample_5,
                R.drawable.sample_6, R.drawable.sample_7 };
    }
```

在主程序 GridViewActivity 中，为 GridView 设置了一个数据适配器，并处理了 GridView 的单击事件。适配器继承自 BaseAdapter 类，与 4.4.15 节中用到的适配器高度相似，在此不再重复。

4.4.17 标签

在有限的手机屏幕空间内，当要浏览的内容较多，无法在一个屏幕空间内全部显示时，可以使用滚动视图来延长屏幕的空间。当浏览的内容具有很强的类别性质时，更合适的方法是将不同类别的内容集中到各自的面板中，这时就需要使用面板标签（Tab）组件了。

Tab 组件利用面板标签把不同的面板内容切换到屏幕上，以显示不同类别的内容。

下面通过一个实例来了解一下 Tab 组件的使用方法。在工程 WidgetDemo 的布局文件 main.xml 中添加一个名为 TabDemo 的 Button，用以启动 TabActivity。

在 main.xml 中添加代码如下：

```
<Button
    android:id="@+id/button13"
    android:layout_width="wrap_content"
    android:layout_height="wrap content"
    android:text="TabDemo" />
```

单击 Button 并启动 GridViewActivity 的代码如下：

```
Button tabbtn=(Button)this.findViewById(R.id.button13);
tabbtn.setOnClickListener(new OnClickListener(){
@Override
public void onClick(View v){
      // TODO Auto-generated method stub
    Intent intent=new Intent(WidgetDemoActivity.this,TabActivity.class);
      startActivity(intent);
}
});
```

同时在 AndroidManifest.xml 文件中声明该 Activity：

```
<activity android:name="TabActivity"></activity>
```

TabActivity 的运行效果如图 4.34 所示。



图 4.34　TabActivity 的运行效果

要使用 Tab 必然涉及它的容器 TabHost，TabHost 包括 TabWigget 和 FrameLayout 两部分。TabWidget 就是每个 Tab 的标签，FrameLayout 是 Tab 的内容。

TabActivity 使用的布局文件是 tab.xml。在 tab.xml 中定义了每个 Tab 中要显示的内容，代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<TabHost xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/tabhost"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
   <LinearLayout
       android:layout_width="fill_parent"
       android:layout_height="fill_parent"
       android:orientation="vertical">
      <TabWidget
          android:id="@android:id/tabs"
          android:layout_width="fill_parent"
          android:layout_height="wrap_content" />
      <FrameLayout
          android:id="@android:id/tabcontent"
          android:layout_width="fill_parent"
          android:layout_height="fill_parent">
         <TextView
             android:id="@+id/tab1"
             android:layout_width="wrap_content"
             android:layout_height="wrap_content"
             android:textSize="40dp"
             android:text="Tab1 页面" />
         <TextView
             android:id="@+id/tab2"
             android:layout_width="wrap_content"
             android:layout_height="wrap_content"
             android:textSize="40dp"
             android:text="Tab2 页面" />
         <TextView
             android:id="@+id/tab3"
             android:layout_width="wrap_content"
             android:layout_height="wrap_content"
             android:textSize="40dp"
             android:text="Tab3 页面" />
      </FrameLayout>
   </LinearLayout>
</TabHost>
```

在 FrameLayout 中我们放置了三个 TextView 组件，分别对应三个 Tab 所显示的内容，当切换不同的 Tab 时会自动显示不同的 TextView 内容。

在主程序 TabActivity 的 OnCreate()方法中，首先获得 TabHost 的对象，并调用 setup()方法进行初始化，然后通过 TabHost.TabSpec 增加 Tab 页，通过 setContent()增加当前 Tab 页显示的内容，通过 setIndicator 增加页的标签，最后设定当前要显示的 Tab 页。

TabActivity 的代码如下：

```
package introduction.android.widgetDemo;

import android.app.Activity;
import android.os.Bundle;
import android.widget.TabHost;

public class TabsActivity extends Activity {
```

```
        public void onCreate (Bundle savedInstanceState) {
            super.onCreate (savedInstanceState) ;
            setContentView (R.layout.tab) ;
            // 步骤 1: 获得 TabHost 的对象, 并进行初始化 setup()
            TabHost tabs= (TabHost) findViewById (R.id.tabhost) ;
            tabs.setup();
            // 步骤 2:通过 TabHost.TabSpec 增加 tab 的一页,通过 setContent()增加内容,通过 setIndicator
增加页的标签
            /* 增加第 1 个 Tab */
            TabHost.TabSpec spec=tabs.newTabSpec ("Tag1") ;
            // 单击 Tab 要显示的内容
            spec.setContent (R.id.tab1) ;
            /* 显示 Tab1 内容 */
            spec.setIndicator ("Tab1") ;
            tabs.addTab (spec) ;
            /* 增加第 2 个 Tab */
            spec=tabs.newTabSpec ("Tag2") ;
            spec.setContent (R.id.tab2) ;// 单击 Tab 要显示的内容
            /* 显示 Tab2 内容 */
            spec.setIndicator ("Tab2") ;
            tabs.addTab (spec) ;
            /* 增加第 3 个 Tab */
            spec=tabs.newTabSpec ("Tag3") ;
            spec.setContent (R.id.tab3) ;// 单击 Tab 要显示的内容
            /* 显示 Tab3 内容 */
            spec.setIndicator ("Tab3") ;
            tabs.addTab (spec) ;
            /* 步骤 3: 可通过 setCurrentTab (index) 指定显示的页, 从 0 开始计算*/
            tabs.setCurrentTab (0) ;
        }
    }
```

除了使用上述方法设置 Tab 页面的显示内容外，还可以使用 setContent（Intent）方法启动某个 Activity，并将该 Activity 的视图作为 Tab 页面的内容。

例如：

```
Intent intent=new Intent().setClass (this, AlbumsActivity.class) ;
spec=tabHost.newTabSpec ("albums") .setIndicator ("Albums",
                    res.getDrawable (R.drawable.ic_tab_albums))
                .setContent (intent) ;
tabHost.addTab (spec) ;
```

4.5　Menu 和 ActionBar

菜单是人机交互的重要接口，在 Android SDK 中，提供了菜单类 android.view.Menu，以完成与菜单有关的操作。

Android SDK 提供三种菜单，分别如下。

- Options Menu：选项菜单，是 Activity 的主要菜单项的集合，当用户单击 Menu 按钮时出现。

在 Android 2.3 以下的版本中，这种菜单最多显示 6 个带图标的菜单项。当菜单中含有 6 个以上的菜单项时，弹出菜单将只显示前 5 个菜单项，第 6 个菜单项会变为 More，单击 More 菜单项后会出现扩展菜单。扩展菜单不支持图标，但支持单选框和复选框。在 Android 3.0（API Level 11）及以上版本中，默认情况下直接弹出的选项菜单不再显示图标。

- Context Menu：上下文菜单，是一个悬浮的菜单项列表，当用户单击注册了上下文菜单的组件时出现。上下文菜单不支持菜单图标和快捷键。
- Submenu：子菜单，是某个菜单项的扩展，是一个悬浮的菜单项列表。子菜单不支持菜单图标或者嵌套子菜单。

4.5.1 Options Menu

要实现选项菜单的功能，首先需要重载 OnCreatOptionsMenu()方法创建菜单，然后通过 onOptionsItemSelected()方法对菜单被单击事件进行监听和处理。

创建一个名为 MenusDemo 的 Android Project，在该工程中对菜单的相关知识进行学习。

在工程的 res 目录下创建一个 menu 目录，用于存放菜单相关的 XML 文件。在该目录下创建 mymenu.xml，代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
  <item
      android:id="@+id/item1"
      android:title="@string/menuitem1"
      android:icon="@drawable/icon01"/>
  <item
      android:id="@+id/item2"
      android:title="@string/menuitem2"
      android:icon="@drawable/icon02"/>
  <item
      android:id="@+id/item3"
      android:title="@string/menuitem3"
      android:icon="@drawable/icon03"/>
  <item
      android:id="@+id/item4"
      android:title="@string/menuitem4"
      android:icon="@drawable/icon04"/>
  <item
      android:id="@+id/item5"
      android:title="@string/menuitem5"
      android:icon="@drawable/icon05"/>
  <item
      android:id="@+id/item6"
      android:title="@string/menuitem6"
      android:icon="@drawable/icon06"/>
  <item
      android:id="@+id/item7"
      android:title="@string/menuitem7"
      android:icon="@drawable/icon07"/>
</menu>
```

mymenu.xml 创建了一个具有 7 个菜单项的菜单，并且通过 android:id 属性为每个菜单项指定 ID，通过 android:title 属性为每个菜单项指定显示的菜单项内容，通过 android:icon 属性指定每个菜单项的图标。对应的图标文件放置到 res/drawable 目录下。

为工程 MenusDemo 创建名为 MenusActivity 的 Activity，将 mymenu.xml 中定义的菜单设置为 MenusActivity 的菜单，重载 OnCreatOptionsMenu()方法。MenusActivity.java 的代码如下：

```
package introduction.android.menusDemo;

import android.app.ActionBar;
import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.View;
import android.widget.TextView;

public class MenusDemoActivity extends Activity {
   private TextView textview;
    /** Called when the activity is first created. */
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
      textview=(TextView)findViewById(R.id.textview1);
   }
   @Override
    public boolean onOptionsItemSelected(MenuItem item) {
         // TODO Auto-generated method stub
    switch(item.getItemId()){
    case R.id.item1:
         textview.setText("item1 selected!");
         break;
    case R.id.item2:
         textview.setText("item2 selected!");
         break;
    case R.id.item3:
         textview.setText("item3 selected!");
         break;
    default:
         break;
    }
         return super.onOptionsItemSelected(item);
    }
    @Override
   public boolean onCreateOptionsMenu(Menu menu) {
      MenuInflater inflater=getMenuInflater();
      inflater.inflate(R.menu.mymenu, menu);
      return true;
   }
}
```

其中：

```
public boolean onCreateOptionsMenu (Menu menu) {
        MenuInflater inflater=getMenuInflater();
        inflater.inflate (R.menu.mymenu, menu) ;
    return true;
}
```

这几行代码通过 MenuInflater. Inflate()方法将 menu.xml 中定义的菜单项内容填充到了菜单中。

在 OnCreatOptionsMenu()方法中创建菜单时也支持 Menu.add()方法，也能达到同样目的，例如：

```
menu.add (0,itemid,0,item_title) ;
```

表示在菜单中添加一个菜单项，该菜单项的 ID 为 itemid，菜单项显示的内容为 item_title 的内容。但是不鼓励使用这种方式，而应该使用 XML 文件来创建菜单。

运行 MenusDemo 实例，单击手机的 Menu 按钮，得到的效果如图 4.35 所示。

由运行效果可见，MenusActivity 已经根据 mymenu.xml 文件创建了一个具有 7 个菜单项的菜单。但是虽然在 mymenu.xml 文件中为每个菜单项指定了一个图标，但是生成的选项菜单中却并没有图标被显示出来，这是为什么呢？

实例 MenusDemo 当前的运行环境是 Android 4.0，其 API Level 为 14。我们先看一下，同样的代码，在 API Level 11 之前的运行效果。

双击打开 AndroidManifest.xml 文件，将其中的代码：

```
<uses-sdk android:minSdkVersion="14" />
```

改为：

```
<uses-sdk android:minSdkVersion="9" />
```

再次运行 MenusDemo 实例，单击 Menu 按钮，得到的效果如图 4.36 所示。



图 4.35 “Menu”按钮运行效果

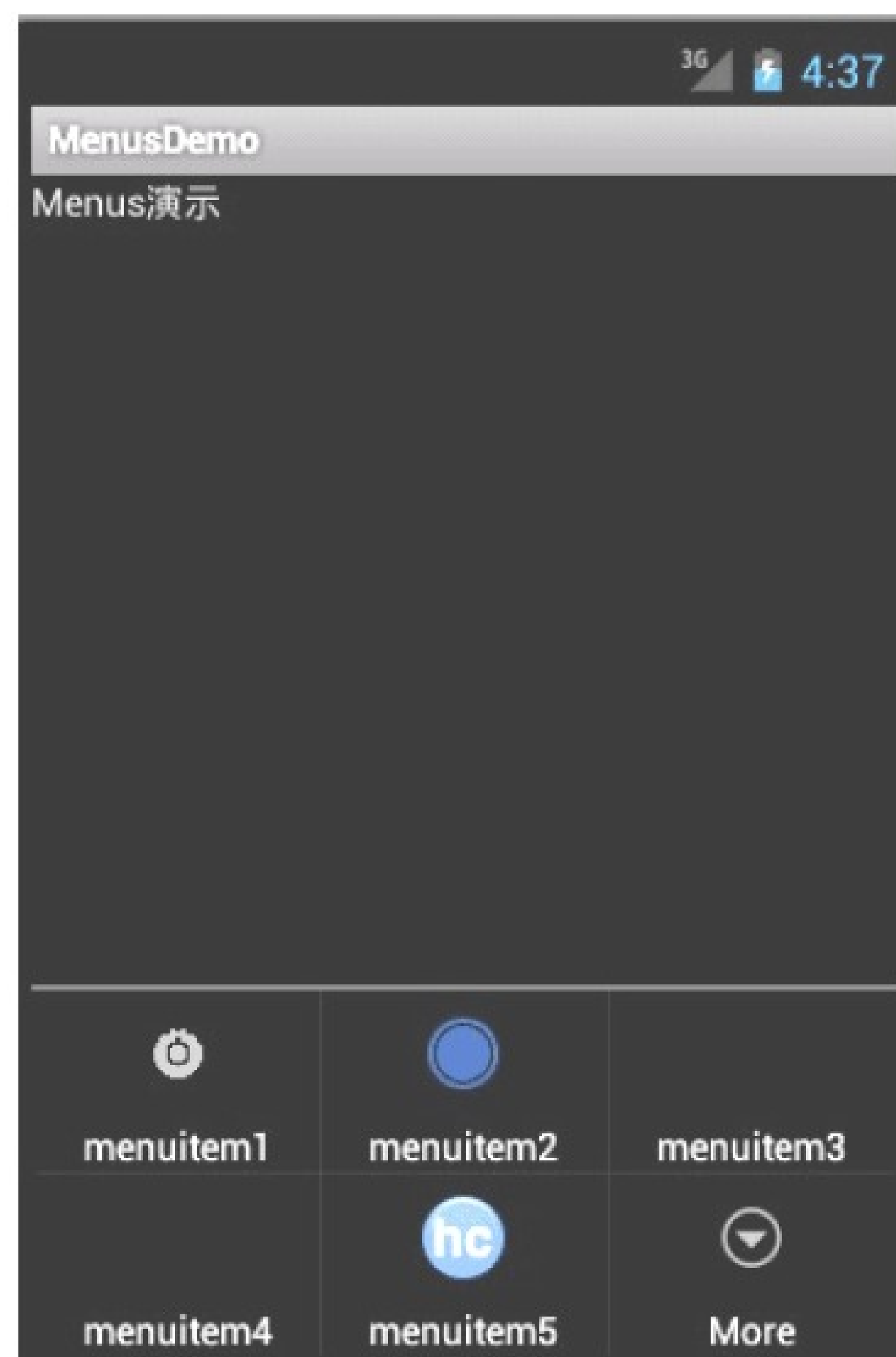


图 4.36 API Level 11 之前 Menu 按钮的运行效果

可见运行在早期的 API 之上的选项菜单效果要更好一些。为什么会出现这种现象呢？其实在 Android SDK 3.0 之后，就不再鼓励直接使用选项菜单，而是将选项菜单和 ActionBar 结合使用。

ActionBar 又称活动栏，位于 Activity 的顶部，取代了原来标题的位置。ActionBar 中包含很多 ActionItem，相当于选项菜单的菜单项。将选项菜单与 ActionBar 结合的方法很简单，只要在 XML 文件中添加一个 android:showAsAction="ifRoom"属性即可。该属性表现如果标题栏有空间的话，就将相关的菜单项放置到 ActionBar 中。如果标题栏空间不足，未能放置到其中的菜单项仍然会以选项菜单的形式出现。ActionBar 的运行效果如图 4.37 所示。



图 4.37　ActionBar 的运行效果

4.5.2　Context Menu

上下文菜单注册到 View 对象上后，用户长按该 View 对象可呼出上下文菜单。上下文菜单悬浮于主界面之上，不支持图标显示和快捷键。其使用方法和选项菜单高度相似，只不过创建上下文菜单的方法为 onCreateContextMenu()，响应上下文菜单单击事件的方法为 onContextItemSelected()。

仍以工程 MenusDemo 为例，为 MenusActivity 的视图中的 TextView 对象添加一个具有两个菜单项的上下文菜单，运行效果如图 4.38 所示。

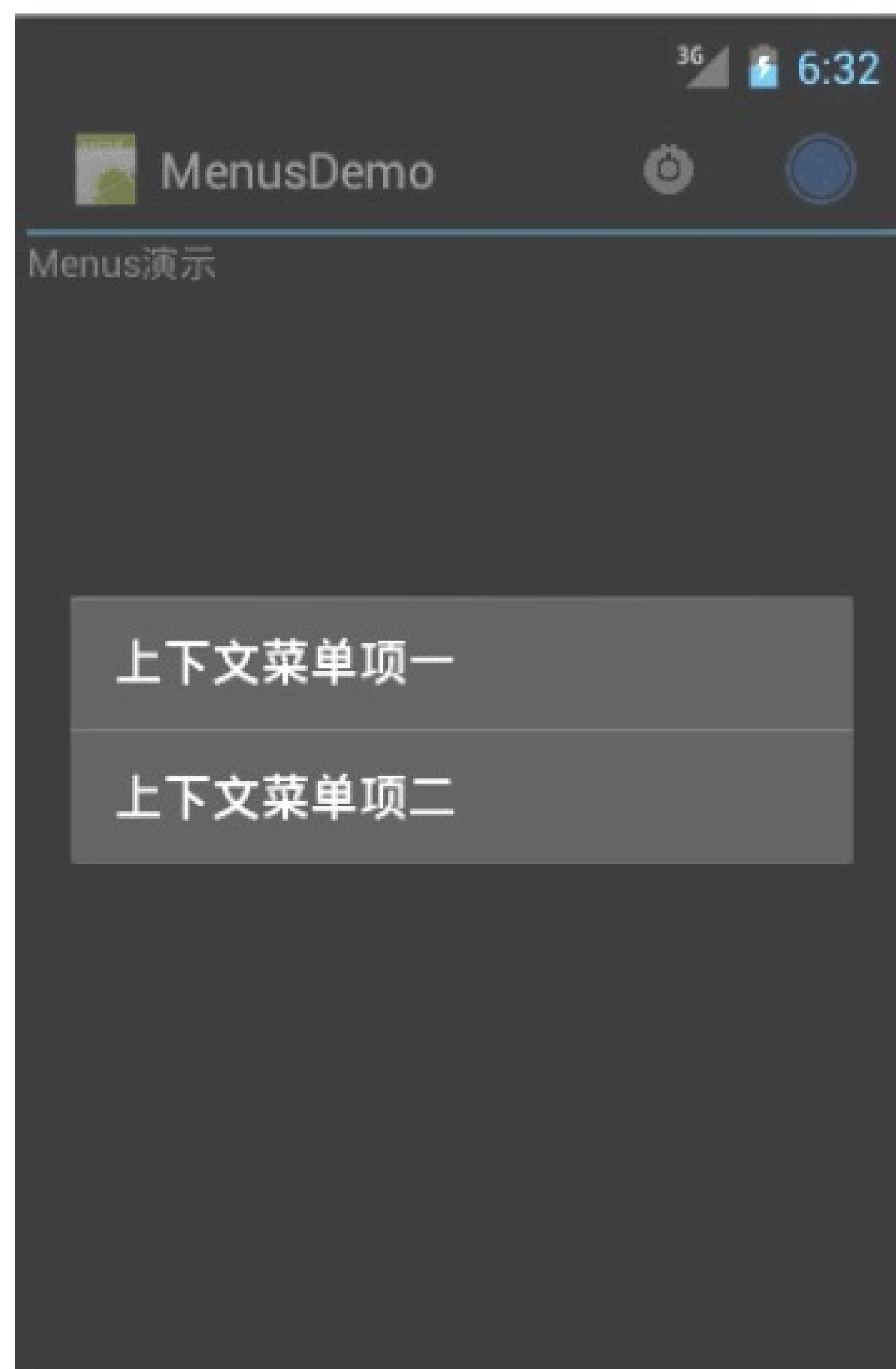


图 4.38　两个上下文菜单的运行结果

为 TextView 对象注册上下文菜单的代码如下：

```
textview=(TextView)findViewById(R.id.textview1);
registerForContextMenu(textview);
```

创建并处理上下文菜单单击事件的代码如下：

```
public boolean onContextItemSelected(MenuItem item){
        // TODO Auto-generated method stub
```

```
            switch (item.getItemId()) {
            case R.id.item6:
                Log.i ("menu","item6!");
                break;
            case R.id.item7:
                Log.i ("menu","item7!");
                break;
            default:
                break;
            }
            return super.onContextItemSelected (item);
        }
        @Override
        public void onCreateContextMenu (ContextMenu menu, View v,
                ContextMenuInfo menuInfo) {
            // TODO Auto-generated method stub
            menu.add (0, R.id.item6, 0, "上下文菜单项一");
            menu.add (0, R.id.item7, 0, "上下文菜单项二");
            super.onCreateContextMenu (menu, v, menuInfo);
        }
```

4.5.3 SubMenu

子菜单可以被添加到其他菜单上，但是子菜单本身不能再有子菜单。使用 addSubMenu()方法为 MenusActivity 的选项菜单添加一个子菜单，运行效果如图 4.39 所示。



图 4.39　添加子菜单的运行效果

实现该子菜单需要重写 onCreateOptionsMenu()方法，代码如下：

```
    public boolean onCreateOptionsMenu (Menu menu) {
            MenuInflater inflater=getMenuInflater();
            inflater.inflate (R.menu.mymenu, menu);
            SubMenu submenu=menu.addSubMenu ("子菜单");
            submenu.add (0,1,0,"子菜单项一");
```

```
        submenu.add (0, 2, 0, "子菜单项二") ;
        return true;
    }
```

子菜单的事件处理代码在 onOptionsItemSelected()中实现。

MenusActivity.java 的完整代码如下：

```
package introduction.android.menusDemo;

import android.app.ActionBar;
import android.app.Activity;
import android.os.Bundle;
import android.util.Log;
import android.view.ContextMenu;
import android.view.ContextMenu.ContextMenuInfo;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.SubMenu;
import android.view.View;
import android.widget.TextView;

public class MenusDemoActivity extends Activity {
    private TextView textview;
     /** Called when the activity is first created. */
    @Override
    public void onCreate (Bundle savedInstanceState) {
        super.onCreate (savedInstanceState) ;
        setContentView (R.layout.main) ;
        textview= (TextView) findViewById (R.id.textview1) ;
        registerForContextMenu (textview) ;
    //   setContentView (textview) ;
    }
    @Override
     public boolean onOptionsItemSelected (MenuItem item) {
          // TODO Auto-generated method stub
     switch (item.getItemId()) {
     case 1:
          Log.i ("menu","submenu item 1 selected") ;
     case R.id.item1:
          textview.setText ("item1 selected!") ;
          break;
     case R.id.item2:
          textview.setText ("item2 selected!") ;
          break;
     case R.id.item3:
          textview.setText ("item3 selected!") ;
          break;
     default:
          Log.i ("menu","other items selected") ;
          break;
     }
          return super.onOptionsItemSelected (item) ;
     }
     @Override
```

```
        public boolean onCreateOptionsMenu (Menu menu) {
            MenuInflater inflater=getMenuInflater();
            inflater.inflate (R.menu.mymenu, menu) ;
            SubMenu submenu=menu.addSubMenu ("子菜单") ;
            submenu.setIcon (android.R.drawable.ic_menu_crop) ;
            submenu.add (0,1,0,"子菜单项一") ;
            submenu.add (0, 2, 0, "子菜单项二") ;
            return true;
        }
         @Override
         public boolean onContextItemSelected (MenuItem item) {
              // TODO Auto-generated method stub

              switch (item.getItemId()) {
              case R.id.item6:
                   Log.i ("menu","item6!") ;
                   break;
              case R.id.item7:
                   Log.i ("menu","item7!") ;
                   break;
              default:
                   break;
              }
              return super.onContextItemSelected (item) ;
         }
         @Override
         public void onCreateContextMenu (ContextMenu menu, View v,
                   ContextMenuInfo menuInfo) {
              // TODO Auto-generated method stub
              menu.add (0, R.id.item6, 0, "上下文菜单项一") ;
              menu.add (0, R.id.item7, 0, "上下文菜单项二") ;
              super.onCreateContextMenu (menu, v, menuInfo) ;
         }

    }
```

4.6 Bitmap

Bitmap 称为点阵图像或绘制图像，是由称作像素（图片元素）的单个点组成的，这些点通过不同的排列和染色以构成图样。Bitmap 是 Android 系统中图像处理最重要的类之一，用它可以获取图像文件信息，对图像进行剪切、旋转、缩放等操作，并可以将图像保存成特定格式的文件。Bitmap 位于 android.graphics 包中，不提供对外的构造方法，只能通过 BitmapFactory 类进行实例化。利用 BitmapFactory 的 decodeFile 方法可以从特定文件中获取 Bitmap 对象，也可以使用 decodeResource()从特定的图片资源中获取 Bitmap 对象。

实例 BitmapDemo 从资源文件中创建 Bitmap 对象，并对其进行一些操作，运行效果如图 4.40 所示。

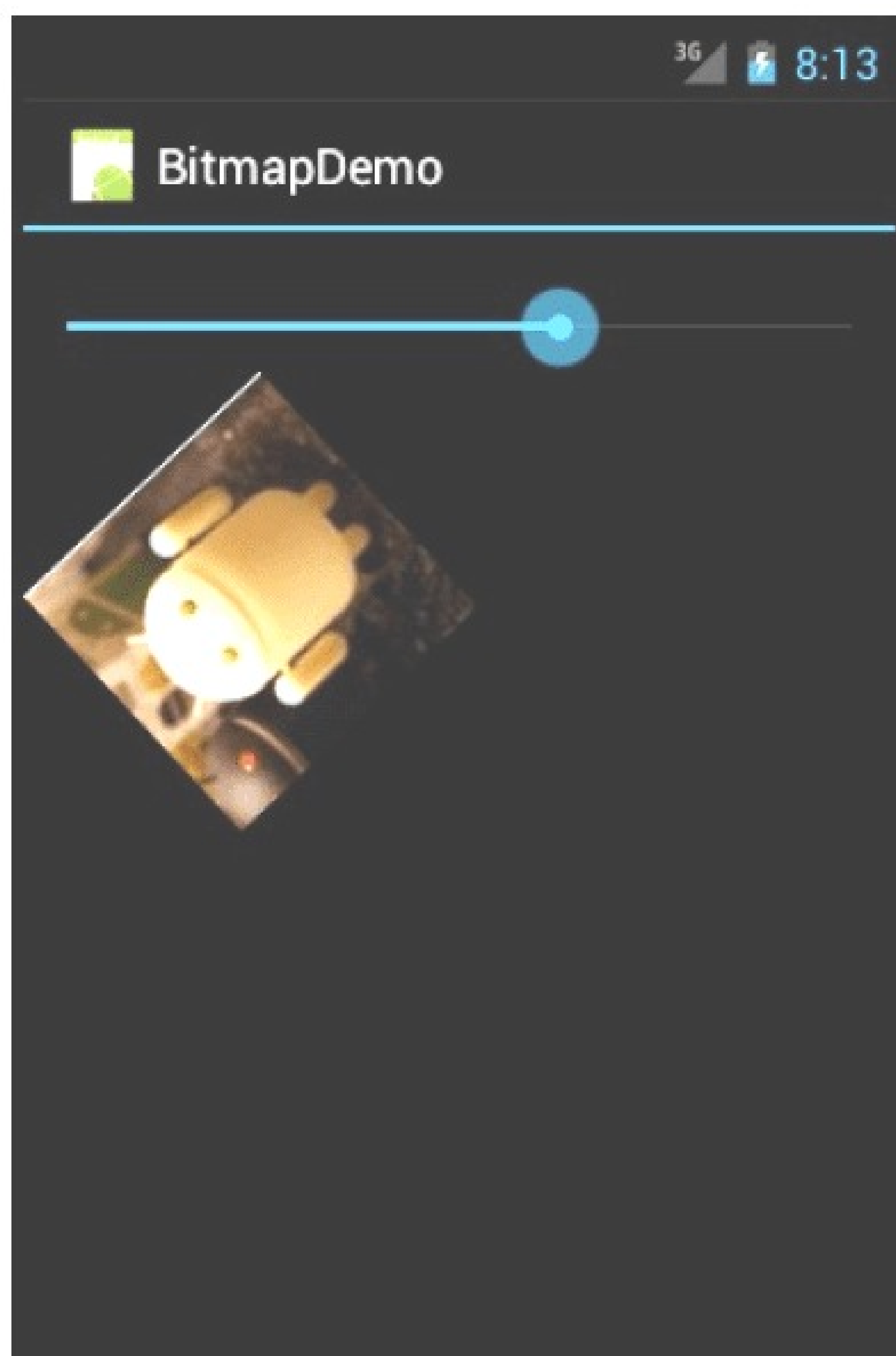


图 4.40　Bitmap 对象的效果

其对应布局文件 Main.xml 的内容如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical">
   <SeekBar
       android:id="@+id/seekBarId"
       android:layout_width="fill_parent"
       android:layout_height="wrap_content" />

   <ImageView
       android:id="@+id/imageview"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:src="@drawable/im01" />

</LinearLayout>
```

BitmapDemoActivity.Java 的代码如下：

```
package introduction.android.bitmapDemo;

import com.sie.bitmapdemo.R;
import android.app.Activity;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.Matrix;
import android.os.Bundle;
import android.widget.ImageView;
import android.widget.SeekBar;
import android.widget.SeekBar.OnSeekBarChangeListener;
```

```
import android.widget.TextView;

public class BitmapDemoActivity extends Activity
{
    ImageView myImageView;
    Bitmap myBmp, newBmp;
    int bmpWidth, bmpHeight;
    SeekBar seekbarRotate;
    float rotAngle;

    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        myImageView=(ImageView) findViewById(R.id.imageview);
        // 由 Resource 载入图片
        myBmp=BitmapFactory.decodeResource(getResources(), R.drawable.im01);
        bmpWidth=myBmp.getWidth();
        bmpHeight=myBmp.getHeight();
        // 实例化 matrix
        Matrix matrix=new Matrix();
        //设定 Matrix 属性 x、y 缩放比例为 1.5
        matrix.postScale(1.5F, 1.5F);
        //顺时针旋转 45 度
        matrix.postRotate(45.0F);
        newBmp=Bitmap.createBitmap(myBmp, 0, 0, bmpWidth, bmpHeight, matrix, true);
        seekbarRotate=(SeekBar) findViewById(R.id.seekBarId);
        seekbarRotate.setOnSeekBarChangeListener(onRotate);
    }
    private SeekBar.OnSeekBarChangeListener onRotate=new SeekBar.OnSeekBarChangeListener(){

        public void onStopTrackingTouch(SeekBar seekBar)
        {
            // TODO Auto-generated method stub

        }

        public void onStartTrackingTouch(SeekBar seekBar)
        {
            // TODO Auto-generated method stub

        }

        public void onProgressChanged(SeekBar seekBar, int progress,
                boolean fromUser)
        {
            // TODO Auto-generated method stub
            Matrix m=new Matrix();
            m.postRotate((float) progress*3.6F);
            newBmp=Bitmap.createBitmap(myBmp, 0, 0, bmpWidth, bmpHeight, m, true);
            myImageView.setImageBitmap(newBmp);
        }
    };
}
```

本实例实现了拖动进度条图片旋转的效果。使用 BitmapFactory 从资源中载入图片，并获取图片的宽和高，之后使用 Matrix 类对图片进行缩放和旋转操作。

4.7 对 话 框

对话框是人机交互过程中十分常见的组件，一般用于在特定条件下对用户显示一些信息，可以增强应用的友好性。

Dialog 类是对话框的基类。对话框虽然可以在界面上显示，但是 Dialog 不是 View 类的子类，而是直接继承自 java.lang.Object 类。Dialog 对象也有自己的生命周期，其生命周期由创建它的 Activity 进行管理。Activity 可以调用 showDialog（int id）将不同 ID 的对话框显示出来，也可以调用 dismissDialog(int id)方法将 ID 标识的对话框从用户界面中关闭掉。当 Activity 调用了 showDialog（ID）方法，对应 ID 的对话框没有被创建时，Android 系统会回调 OnCreateDialog（ID）方法来创建具有该 ID 的对话框。在 Activity 中创建的对话框都会被 Activity 保存，下次 showDialog（ID）方法被调用时，若该 ID 的对话框已经被创建，则系统不会再次调用 OnCreateDialog（ID）方法创建该对话框，而是会回调 onPrepareDialog（int id, Dialog dialog）方法，该方法允许对话框在被显示之前做一些修改。

常用的对话框有 AlertDialog 和 ProgressDialog，本节将通过实例讲解这两种对话框的使用方法。

4.7.1 AlertDialog

AlertDialog 对话框是十分常用的用于显示信息的方式，最多可提供三个按钮。AlertDialog 不能直接通过构造方法构建，而要由 AlertDialog.Builder 类来创建。AlertDialog 对话框的标题、按钮以及按钮要响应的事件也由 AlertDialog.Builder 设置。

在使用 AlertDialog. Builder 创建对话框时常用的几个方法如下。

- setTitle(): 设置对话框中的标题。
- setIcon(): 设置对话框中的图标。
- setMessage(): 设置对话框的提示信息。
- setPositiveButton(): 为对话框添加 yes 按钮。
- setNegativeButton(): 为对话框添加 no 按钮。
- setNeutralButton(): 为对话框添加第三个按钮。

下面通过实例来学习创建 AlertDialog 的方法。

创建 Android 工程 DialogDemo，并在 main.xml 中添加两个按钮，分别为 AlertDialog 和 ProcessDialog。

其 main.xml 代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
```

```
    android:layout_height="fill_parent"
    android:orientation="vertical">

   <TextView
       android:layout_width="fill parent"
       android:layout_height="wrap content"
       android:text="Dialog 演示" />

   <Button
       android:id="@+id/button1"
       android:layout_width="match parent"
       android:layout_height="wrap_content"
       android:text="AlertDialog" />

   <Button
       android:id="@+id/button2"
       android:layout_width="match parent"
       android:layout_height="wrap_content"
       android:text="ProgressDialog" />

</LinearLayout>
```

其运行效果如图 4.41 所示。

处理 AlertDialog 按钮单击事件的代码为:

```
btn=(Button)findViewById(R.id.button1);
        btn.setOnClickListener(new OnClickListener(){
                @Override
                public void onClick(View v){
                     // TODO Auto-generated method stub
                     showDialog(ALERT_DLG);
                }
        });
```

单击 AlertDialog 按钮，调用 showDialog（ALERT_DLG），系统回调 onCreateDialog（int id）方法，创建并弹出 AlertDialog 对话框，如图 4.42 所示。



图 4.41　AlertDialog 的运行效果

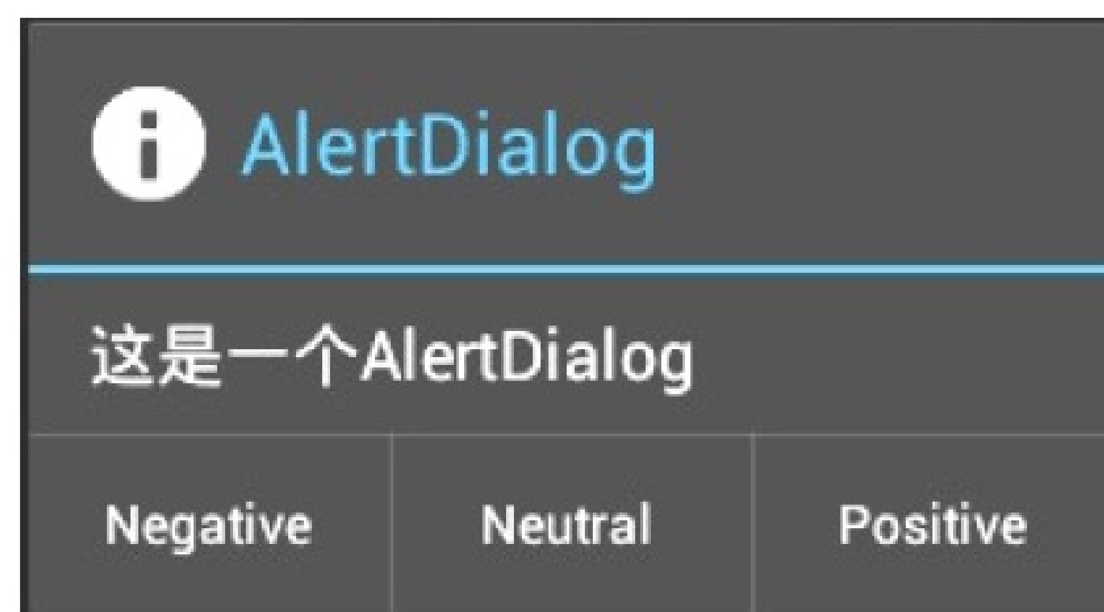


图 4.42　单击 AlertDialog 按钮的效果

相关代码为：

```
protected Dialog onCreateDialog (int id) {
        // TODO Auto-generated method stub
        Dialog dialog=null;
        switch (id) {
        case ALERT_DLG:
            AlertDialog.Builder builder=new AlertDialog.Builder(DialogDemoActivity.this);
            builder.setIcon (android.R.drawable.ic_dialog_info) ;
            builder.setTitle ("AlertDialog") ;
            builder.setMessage ("这是一个 AlertDialog") ;
            builder.setPositiveButton ("Positive",new DialogInterface.OnClickListener(){

                @Override
                public void onClick (DialogInterface dialog, int which) {
                    // TODO Auto-generated method stub
                    Log.i ("DialogDemo","OK 按钮被单击! ") ;
                }

            }) ;
            builder.setNegativeButton ("Negative",new DialogInterface.OnClickListener(){

                @Override
                public void onClick (DialogInterface dialog, int which) {
                    // TODO Auto-generated method stub
                    Log.i ("DialogDemo","Cancel 按钮被单击! ") ;
                }

            }) ;
            builder.setNeutralButton ("Neutral",new DialogInterface.OnClickListener(){

                @Override
                public void onClick (DialogInterface dialog, int which) {
                    // TODO Auto-generated method stub
                    Log.i ("DialogDemo","Neutral 按钮被单击! ") ;
                }

            }) ;
            dialog=builder.create();
            break;
        default:
            break;
        }
        return dialog;
}
```

onCreateDialog()方法中创建了带有三个按钮的 AlertDialog，并且为每个按钮添加了事件处理方法，以便获知用户单击了哪个按钮。

4.7.2　ProgressDialog

ProgressDialog 是一个带有进度条的对话框，当应用程序在完成比较耗时的工作时，使用该对话框可以为用户提供一个总进度上的提示。

为 main.xml 布局中的 ProgressDialog 按钮添加事件处理代码：

```
progressbtn=(Button)findViewById(R.id.button2);
    progressbtn.setOnClickListener(new OnClickListener(){

        @Override
        public void onClick(View v){
            // TODO Auto-generated method stub
            showDialog(PROGRESS_DLG);
        }

    });
```

单击 ProgressDialog 按钮，调用 showDialog(PROGRESS_DLG)，系统回调 onCreateDialog(int id) 方法，创建并弹出 ProgressDialog 对话框，如图 4.43 所示。

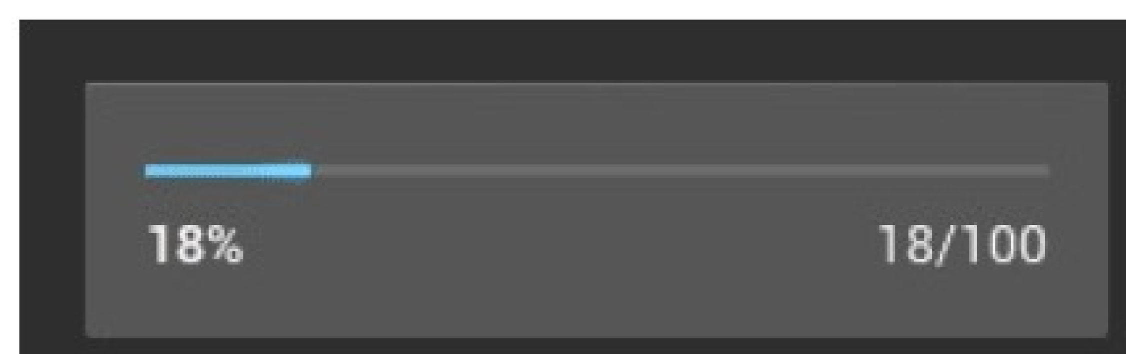


图 4.43　单击 ProgressDialog 按钮的效果

onCreateDialog()方法中的相关代码如下：

```
case PROGRESS_DLG:
            progressDialog=new ProgressDialog(this);
            //设置水平进度条
            progressDialog.setProgressStyle(progressDialog.STYLE_HORIZONTAL);
            //设置进度条最大值为 100
            progressDialog.setMax(100);
            //设置进度条当前值为 0
            progressDialog.setProgress(0);
            dialog=progressDialog;
            new Thread(new Runnable(){
                int count=0;
                @Override
                public void run(){
                    // TODO Auto-generated method stub
                    while(progressDialog.getProgress()<100){
                        count+=3;
                        progressDialog.setProgress(count);
                        try {
                            Thread.sleep(1000);
                        } catch(InterruptedException e){
                            // TODO Auto-generated catch block
                            e.printStackTrace();
                        }
                    }
                }

            }).start();
        break;
```

4.8　Toast 和 Notification

Toast 和 Notification 是 Android 系统为用户提供的轻量级的信息提醒机制。这种方式不会打断用户当前的操作，也不会获取到焦点，非常方便。

本节我们通过实例学习 Toast 和 Notification 的使用方法。

4.8.1　Toast

创建工程 NotificationDemo，并实现如图 4.44 所示的布局。

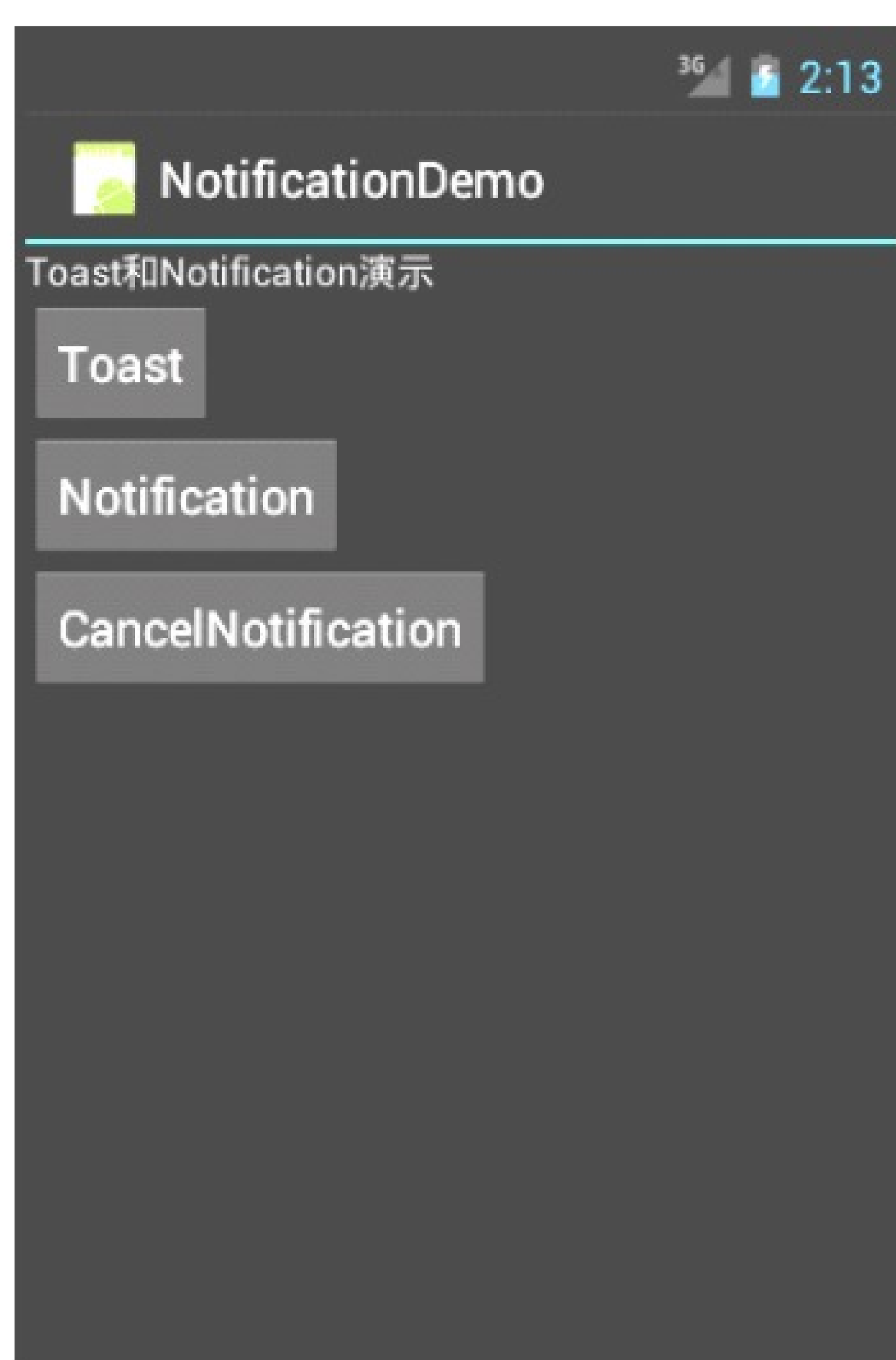


图 4.44　工程布局

main.xml 的代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical">

   <TextView
       android:layout_width="fill_parent"
       android:layout_height="wrap_content"
       android:text="Toast 和 Notification 演示" />

   <Button
       android:id="@+id/button1"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:text="Toast" />

   <Button
```

```
        android:id="@+id/button2"
        android:layout_width="wrap content"
        android:layout_height="wrap content"
        android:text="Notification" />

    <Button
        android:id="@+id/button3"
        android:layout_width="wrap content"
        android:layout_height="wrap content"
        android:text="CancelNotification" />

</LinearLayout>
```

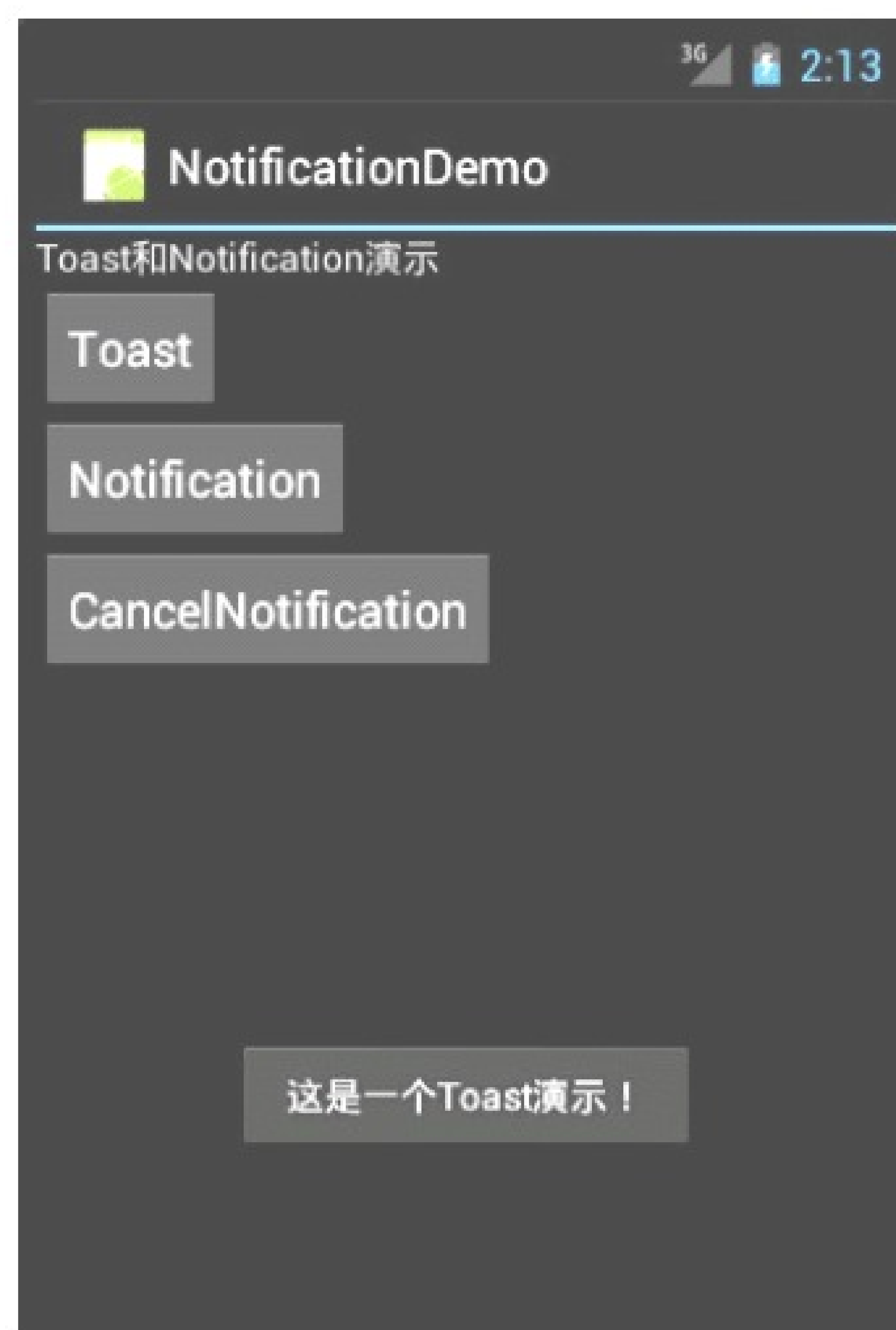


图 4.45　单击 Toast 按钮的效果

在 NotificationDemoActivity 中为每个按钮添加事件响应。单击 Toast 按钮，运行效果如图 4.45 所示。

相关代码如下：

```
Button toastBtn= (Button) this.findViewById
(R.id.button1) ;
        toastBtn.setOnClickListener (new
View.OnClickListener(){
                @Override
                public void onClick (View v) {
                    // TODO Auto-generated method stub
                    Toast.makeText (NotificationDemoActivity.this, "这是一个 Toast 演示! ",
Toast.LENGTH_LONG) .show();
                }
            }) ;
```

Toast 用于向用户显示小信息量的提示，它不会中断应用程序进程，不会对用户操作造成任何干扰，也不能与用户交互，在信息显示后会自动消失。此处使用 Toast.makeText（Context context, CharSequence text, int duration）方法来创建一个 Toast。其中，context 指显示 Toast 的上下文；text 指 Toast 中显示的文字内容；duration 指 Toast 显示延续的时间，该时间可以直接指定，也可以使用 Toast 提供 LENGTH_LONG 和 LENGTH_SHORT 常量。Toast.show()方法可以将 Toast 对象显示出来。Toast 默认情况下显示在屏幕的下方，可以通过 Toast.setGravity()方法设置 Toast 的显示位置。例如如下代码：

```
Toast toast=Toast.makeText (NotificationDemoActivity.this,
                        "这是一个位于中间位置的 Toast",
Toast.LENGTH_LONG) ;
                toast.setGravity (Gravity.CENTER, 0, 0) ;
            toast.show();
```

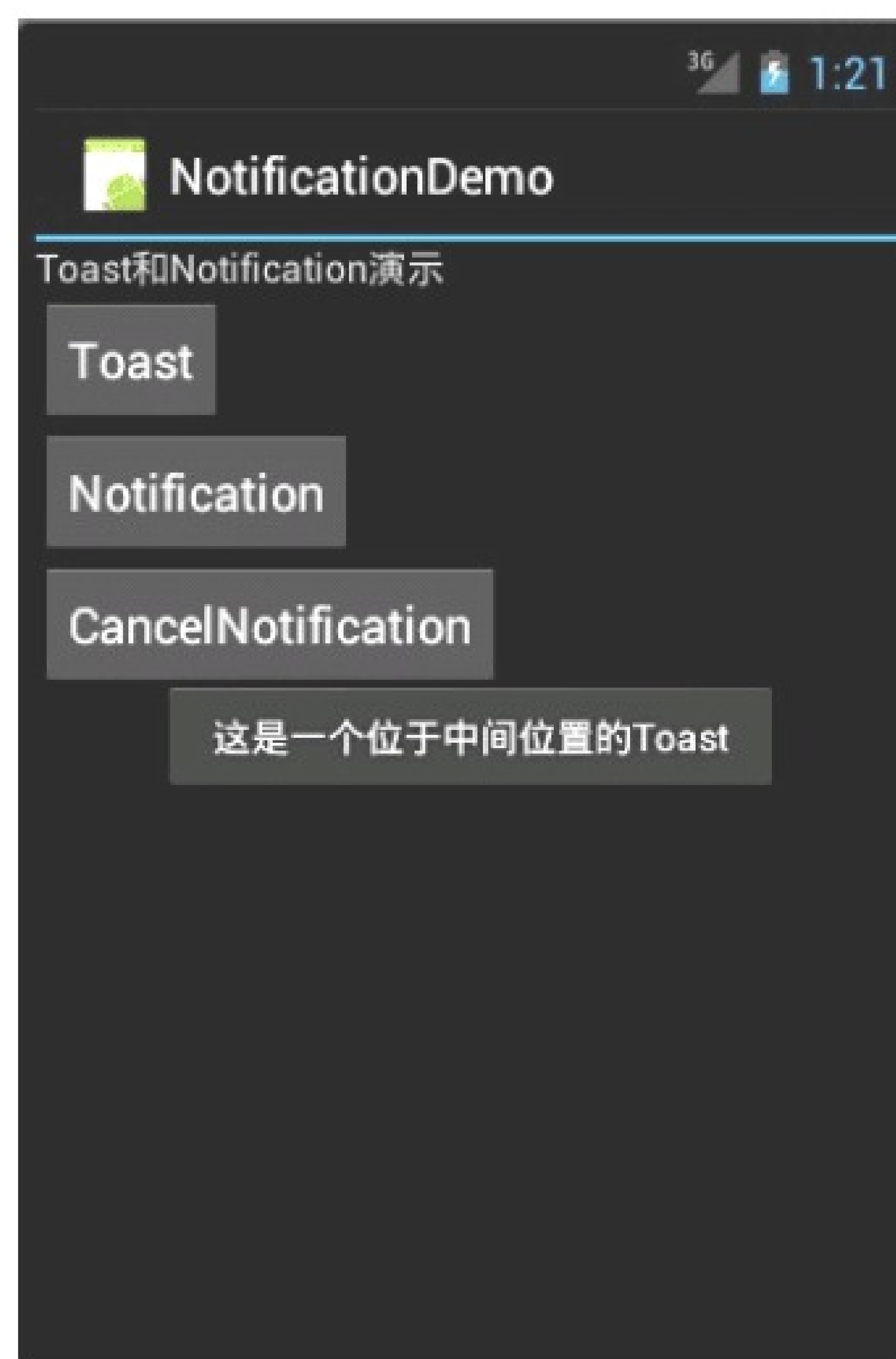


图 4.46　显示效果

显示效果如图 4.46 所示。

4.8.2　Notification

Notification 可以在手机屏幕顶部的状态栏显示一个带图标的通知，同时播放声音或者使手机

震动。Notification 可以扩展以显示详细信息，单击该 Notification 还可以跳转到特定的 Activity。

单击 Notification 按钮，运行效果如图 4.47 所示，在视图的状态栏出现 Notification 提示。按住 Notification 并下拉，可将 Notification 内容进行扩展，效果如图 4.48 所示。单击图标处，应用程序跳转到 NoteActivity 视图，运行效果如图 4.49 所示。单击“返回”按钮，返回到 NotificationDemoActivity 视图。

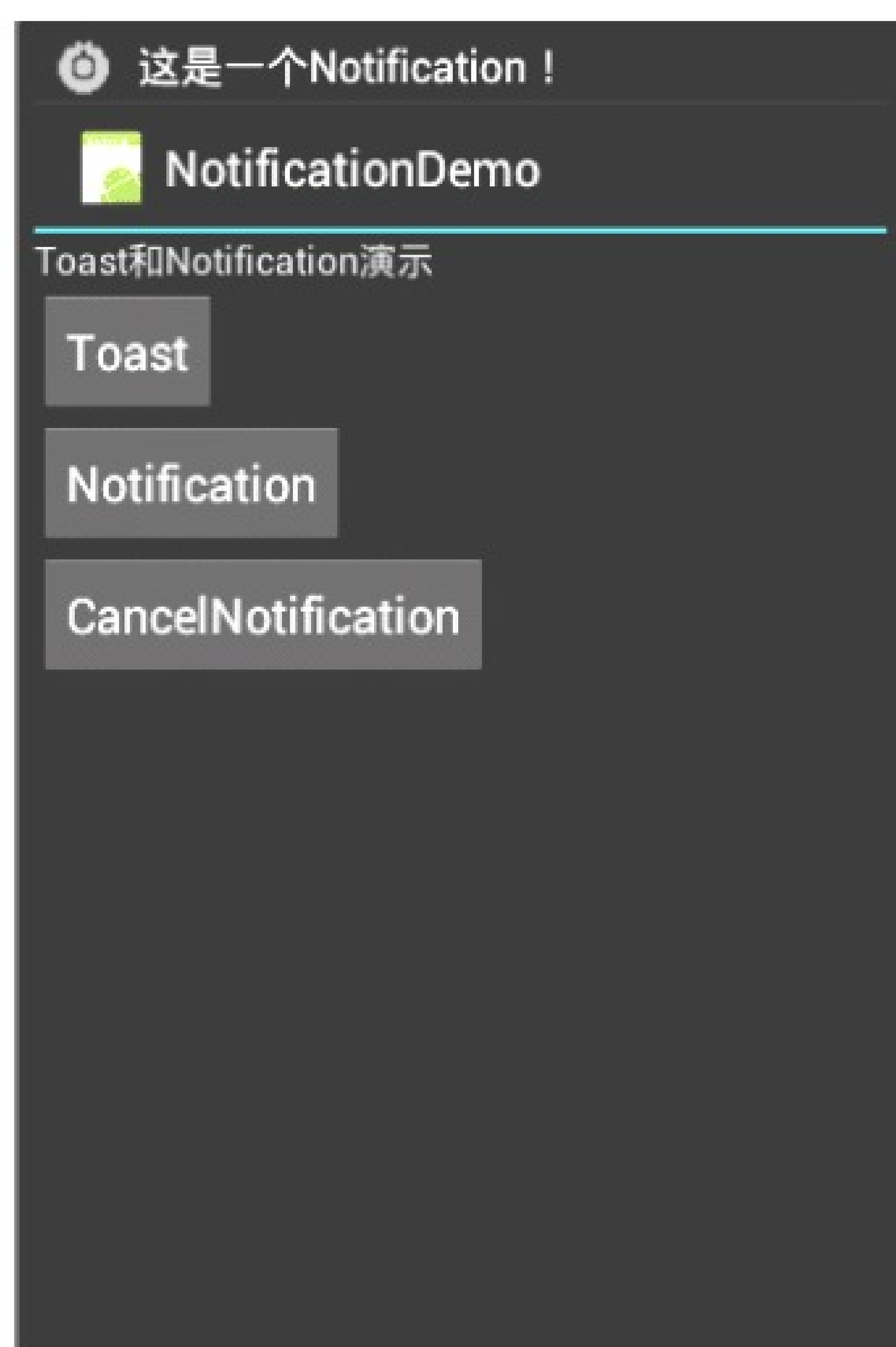


图 4.47　单击 Notification 按钮的效果

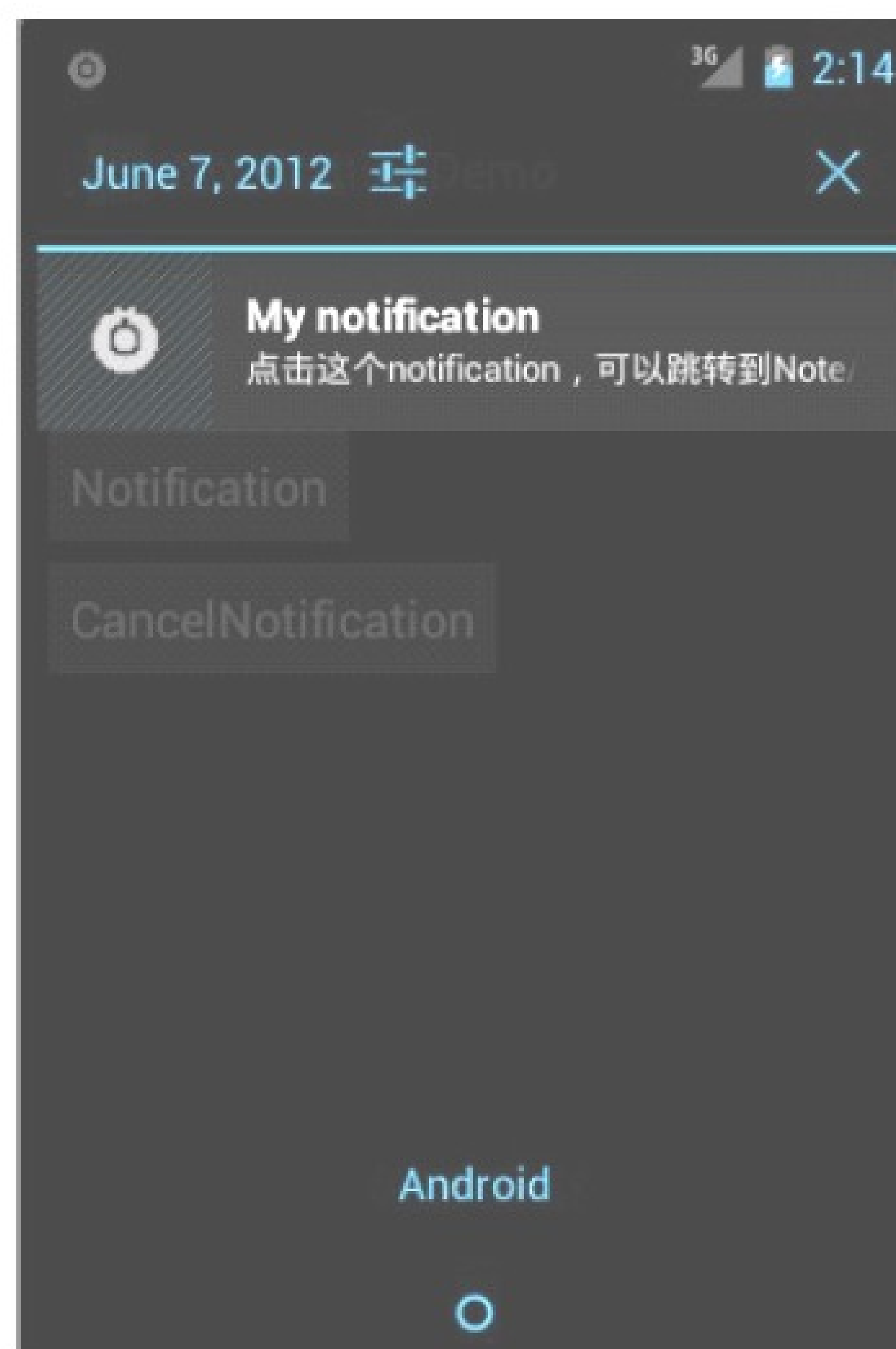


图 4.48　下拉 Notification 的效果

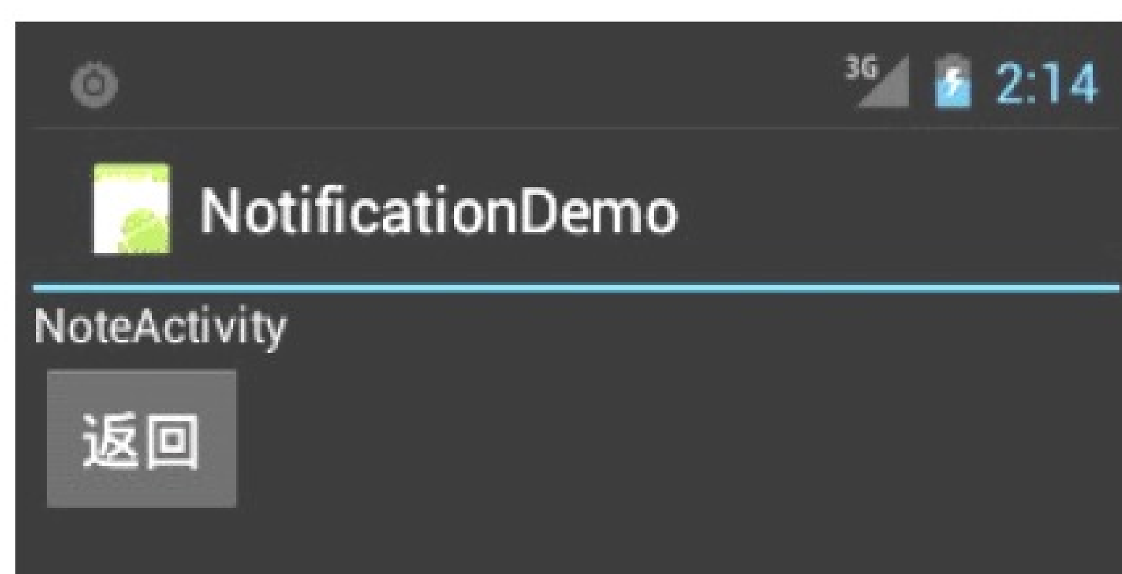


图 4.49　单击图标的效果

相关代码如下：

```
Button notifyBtn=（Button）this.findViewById（R.id.button2）；
    notifyBtn.setOnClickListener（new View.OnClickListener(){

        @Override
        public void onClick（View v）{
            // TODO Auto-generated method stub
            context=getApplicationContext();
            String ns=Context.NOTIFICATION_SERVICE;
            mNotificationManager=（NotificationManager）getSystemService（ns）；

            int icon=R.drawable.icon01;
            CharSequence tickerText="这是一个Notification！";
            long when=System.currentTimeMillis();
            Notification.Builder builder=new Notification.Builder（context）；
            builder.setSmallIcon（icon）；
            builder.setTicker（tickerText）；
```

```
                    builder.setWhen (when) ;
                    notification=builder.getNotification();
                    CharSequence contentTitle="My notification";
                    CharSequence contentText="单击这个notification，可以跳转到NoteActivity.";
                    Intent notificationIntent=new Intent (context, NoteActivity.class) ;
                    PendingIntent contentIntent=PendingIntent.getActivity (context, 0,
notificationIntent, 0) ;
                    notification.setLatestEventInfo (context, contentTitle, contentText,
contentIntent) ;
                    notification.defaults=notification.DEFAULT_SOUND;
                    mNotificationManager.notify (NOTIFICATION_ID, notification) ;
               }
     }) ;
```

Notification.Builder 是 Android API Level 11 以上版本提供的 Notification 的创建类，可以方便地创建 Notification 并设置各种属性。此处创建了一个 Notification，并指定了显示内容和图标。Notification.setLatestEventInfo()方法设定了当用户扩展 Notification 时显示的样式，并通过 PendingIntent 对象指定了当用户单击扩展的 Notification 时应用程序如何跳转，此处跳转至 NoteActivity。NotificationManager.notify（int id，Notification notification）方法为 Notification 对象指定一个 ID 值，并将该 Notification 对象显示到状态栏上。NotificationManager.cancel（int id）方法会将 ID 指向的 Notification 对象取消掉。

NoteActivity.java 的代码如下：

```
     package introduction.android.notificationDemo;

     import android.app.Activity;
     import android.content.Intent;
     import android.os.Bundle;
     import android.view.View;
     import android.widget.Button;

     public class NoteActivity extends Activity {
          private Button btn;

         public void onCreate (Bundle savedInstanceState) {
                super.onCreate (savedInstanceState) ;
                setContentView (R.layout.other) ;
                btn= (Button) this.findViewById (R.id.button1) ;
                btn.setOnClickListener (new View.OnClickListener(){

                       @Override
                       public void onClick (View v) {
                            // TODO Auto-generated method stub
                            Intent intent=new Intent
(NoteActivity.this,NotificationDemoActivity.class) ;
                            startActivity (intent) ;
                       }
                  }) ;
          }
     }
```

NoteActivity 所使用的布局文件 other.xml 的代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

   <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="NoteActivity" />

   <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="返回" />

</LinearLayout>
```

4.8.3　Notification Group

当一个应用程序产生多个通知时，Android N 提供了新的 API，支持将多个通知进行分组和折叠显示，同时告诉用户共有多少个通知，并且给出一个关于通知的摘要消息。实例 NotiDemo 演示了这一功能，其界面很简单，布局如图 4.50 所示。当每次点击 NOTIFY 按钮时，该应用会产生一个通知消息，而按钮下方的 TextView 会显示当前应用共产生了多少个通知。

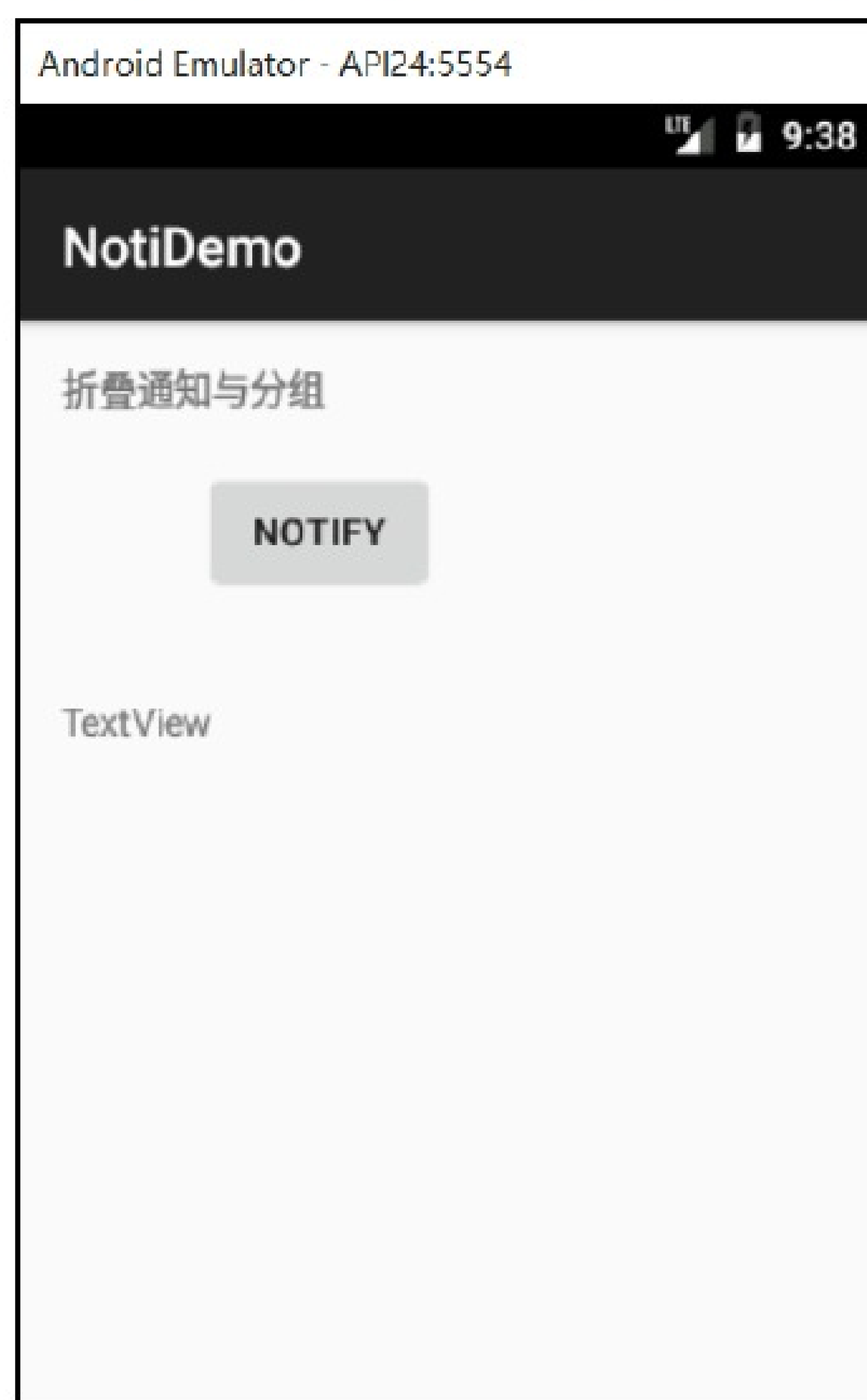


图 4.50　NotiDemo 布局

该布局对应内容为：

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
```

```
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="introduction.android.notidemo.MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!" />

    <Button
        android:text="Notify"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/textView"
        android:layout_alignParentStart="true"
        android:layout_marginStart="52dp"
        android:layout_marginTop="38dp"
        android:id="@+id/button" />

    <TextView
        android:text="TextView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="36dp"
        android:id="@+id/number_of_notifications"
        android:layout_below="@+id/button"
        android:layout_centerHorizontal="true" />
</RelativeLayout>
```

MainActivity.java 的代码为：

```
package introduction.android.notidemo;

import android.app.Activity;
import android.app.Notification;
import android.app.NotificationManager;
import android.content.Context;
import android.os.Bundle;
import android.service.notification.StatusBarNotification;
import android.support.v4.app.NotificationCompat;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

public class MainActivity extends Activity {

    private static final int REQUEST_CODE = 2323;
    private static final String TAG = "NotiDemo";
    private static final String NOTIFICATION_GROUP ="intoduction.android.notidemo.group";
```

```
private static final int NOTIFICATION_GROUP_SUMMARY_ID = 1;
private TextView mNumberOfNotifications;
private NotificationManager mNotificationManager;
private static int sNotificationId = NOTIFICATION_GROUP_SUMMARY_ID + 1;
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    mNotificationManager = (NotificationManager) getSystemService(
            Context.NOTIFICATION_SERVICE);
    mNumberOfNotifications = (TextView)findViewById(R.id.number_of_notifications);
    Button btn= (Button) findViewById(R.id.button);
    btn.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            addNotificationAndUpdateSummaries();
        }
    });
}
private void addNotificationAndUpdateSummaries() {
    // [BEGIN create_notification]
    // Create a Notification and notify the system.
    final NotificationCompat.Builder builder = new NotificationCompat.Builder(this)
            .setSmallIcon(R.mipmap.ic_notification)
            .setContentTitle(getString(R.string.app_name))
            .setContentText(getString(R.string.sample_notification_content))
            .setAutoCancel(true)
            .setGroup(NOTIFICATION_GROUP);

    final Notification notification = builder.build();
    mNotificationManager.notify(getNewNotificationId(), notification);
    // [END create_notification]
    Log.i(TAG, "Add a notification");
    updateNotificationSummary();
    updateNumberOfNotifications();
}

/**
 * Adds/updates/removes the notification summary as necessary.
 */
protected void updateNotificationSummary() {
    int numberOfNotifications = getNumberOfNotifications();

    if (numberOfNotifications > 1) {
        // Add/update the notification summary.
        String notificationContent = getString(R.string.sample_notification_summary_content,
                numberOfNotifications);
        final NotificationCompat.Builder builder = new NotificationCompat.Builder(this)
                .setSmallIcon(R.mipmap.ic_notification)
                .setStyle(new NotificationCompat.BigTextStyle()
                        .setSummaryText(notificationContent))
                .setGroup(NOTIFICATION_GROUP)
                .setGroupSummary(true);
        final Notification notification = builder.build();
        mNotificationManager.notify(NOTIFICATION_GROUP_SUMMARY_ID, notification);
```

```
        } else {
            // Remove the notification summary.
            mNotificationManager.cancel(NOTIFICATION_GROUP_SUMMARY_ID);
        }
    }

    /**
     * Requests the current number of notifications from the {@link NotificationManager} and
     * display them to the user.
     */
    protected void updateNumberOfNotifications() {
        final int numberOfNotifications = getNumberOfNotifications();
        mNumberOfNotifications.setText(getString(R.string.active_notifications,
                numberOfNotifications));
        Log.i(TAG, getString(R.string.active_notifications, numberOfNotifications));
    }

    /**
     * Retrieves a unique notification ID.
     */
    public int getNewNotificationId() {
        int notificationId = sNotificationId++;

        // Unlikely in the sample, but the int will overflow if used enough so we skip the summary
        // ID. Most apps will prefer a more deterministic way of identifying an ID such as hashing
        // the content of the notification.
        if (notificationId == NOTIFICATION_GROUP_SUMMARY_ID) {
            notificationId = sNotificationId++;
        }
        return notificationId;
    }

    private int getNumberOfNotifications() {
        // [BEGIN get_active_notifications]
        // Query the currently displayed notifications.
        final StatusBarNotification[] activeNotifications = mNotificationManager
                .getActiveNotifications();
        // [END get_active_notifications]

        // Since the notifications might include a summary notification remove it from the count if
        // it is present.
        for (StatusBarNotification notification : activeNotifications) {
            if (notification.getId() == NOTIFICATION_GROUP_SUMMARY_ID) {
                return activeNotifications.length - 1;
            }
        }
        return activeNotifications.length;
    }
}
```

对应的 strings.xml 代码为：

```
<resources>
 <string name="app_name">NotiDemo</string>
 <string name="active_notifications">目前的通知数目：%1$d</string>
```

```
 <string name="sample_notification_content">这是一个通知的示例。</string>
 <string name="sample_notification_summary_content">共有 %d 个通知。</string>
</resources>
```

点击 NOTIFY 按钮，运行效果如图 4.51 所示。



图 4.51　运行效果

Android N 通过 NotificationCompat 类构建通知的模板信息，例如通知的图标、通知的标题、通知的内容、通知是否需要进行分组等，然后由 NotificationCompat 构建 Notification 通知对象，并由 NotificationManager 发送通知。相关代码如下：

```
final NotificationCompat.Builder builder = new NotificationCompat.Builder(this)
             .setSmallIcon(R.mipmap.ic_notification)
             .setContentTitle(getString(R.string.app_name))
             .setContentText(getString(R.string.sample_notification_content))
             .setAutoCancel(true)
             .setGroup(NOTIFICATION_GROUP);
final Notification notification = builder.build();
      mNotificationManager.notify(getNewNotificationId(), notification);
```

在设置了通知分组的情况下，Android N 会自动将同一个应用的通知进行合并分组实现，Android N 可以通过 NotificationCompat 设置通知分组的显示消息。

```
String notificationContent = getString(R.string.sample_notification_summary_content,
                numberOfNotifications);
          final NotificationCompat.Builder builder = new NotificationCompat.Builder(this)
                .setSmallIcon(R.mipmap.ic_notification)
                .setStyle(new NotificationCompat.BigTextStyle()
                       .setSummaryText(notificationContent))
                .setGroup(NOTIFICATION_GROUP)
                .setGroupSummary(true);
```

默认情况下，通知栏会分别显示每条通知。当产生的通知数目较多时，之前的通知会被折叠，并以“+折叠通知数目”的方式进行显示，如图 4.52 所示。

将折叠效果下的通知分组下拉，会得到非折叠效果的通知列表，如图 4.53 所示。而这也是不进行通知分组折叠时的效果，即 NotificationCompat 不进行 setGroup 设置时的效果。

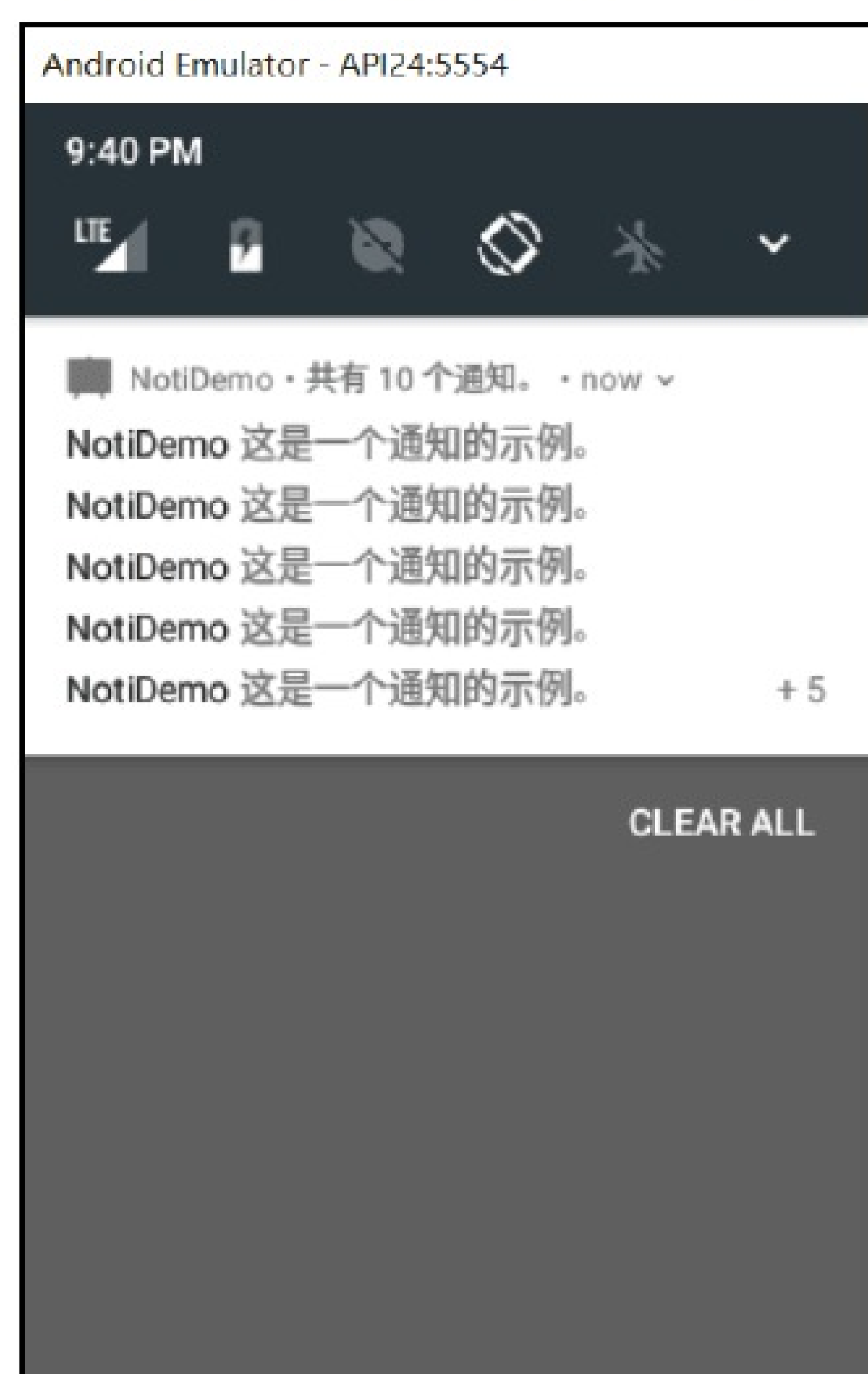


图 4.52　通知分组和折叠效果



图 4.53　非折叠的通知列表

4.9　多窗口模式

Android N 支持多窗口模式，或者叫分屏模式，即在屏幕上可以同时显示多个窗口。在手机模式下，两个应用可以并排或者上下同时显示，如图 4.54 所示，屏幕上半部分的窗口是系统的 CLOCK 应用，下半部分是系统设置功能。用户可以拖动两个应用之间的分界线改变两个窗口的大小，放大其中一个应用，同时缩小另一个应用。在电视设备上，可以实现“画中画”功能。在分屏模式下，各个窗口的应用都可以正常运行，但是只能有一个窗口获得焦点，而另外的窗口则属于暂停状态。

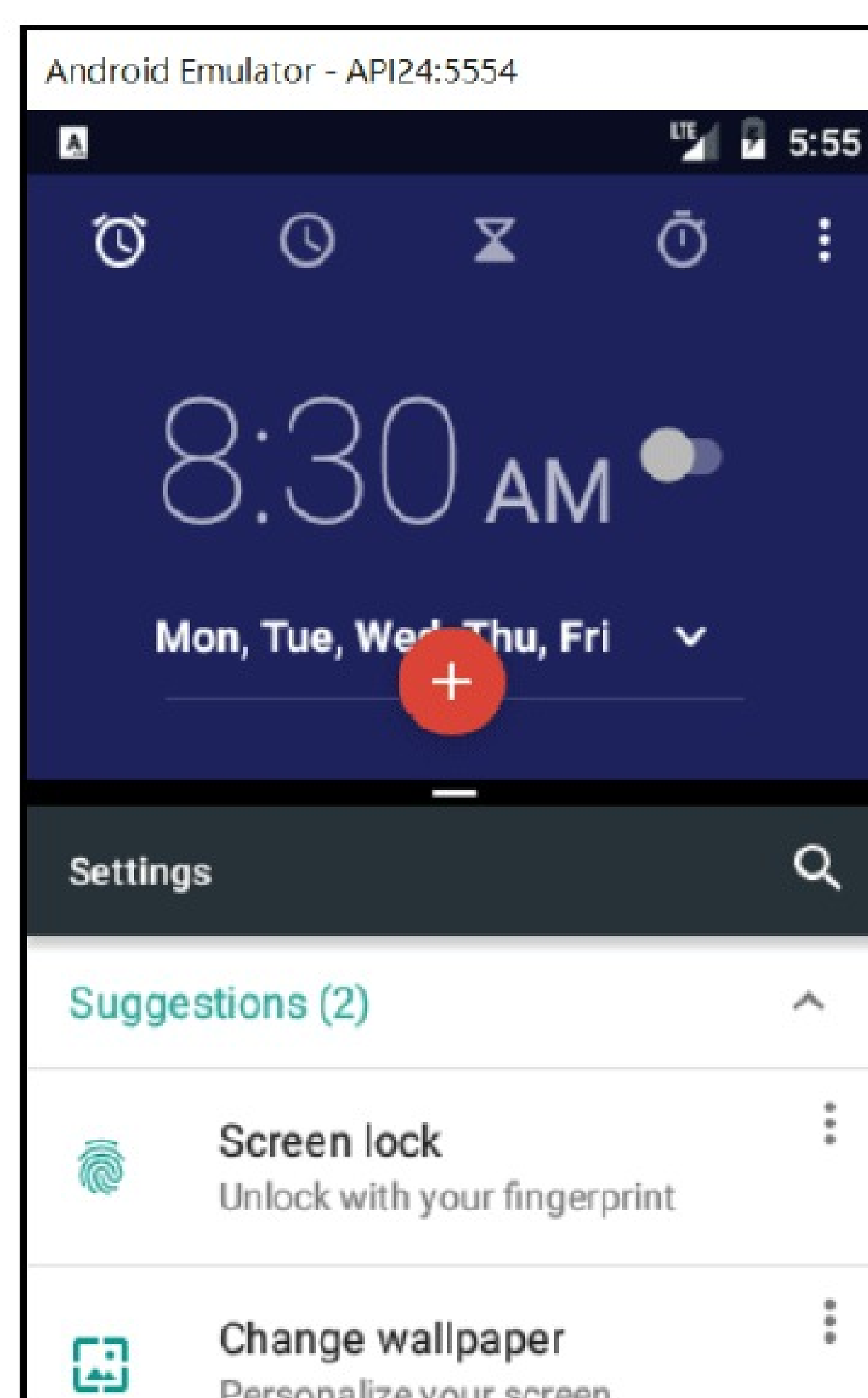


图 4.54　分屏模式

Android N 用户可以通过以下方式切换到多窗口模式：

（1）用户打开 Overview 屏幕并长按 Activity 标题，可以拖动该 Activity 至屏幕突出显示的区域，使 Activity 进入多窗口模式。

（2）用户长按 Overview 按钮，设备上当前的 Activity 将进入多窗口模式，同时将打开 Overview 屏幕，用户可在该屏幕中选择要共享屏幕的另一个 Activity。

用户可以在两个 Activity 共享屏幕的同时在这两个 Activity 之间拖放数据。

默认情况下，Android N 的 Activity 都是开启多窗口模式的。例如，我们通过 Android Studio 构建一个默认的空 Activity 应用 MultiScreenDemo，无须做任何修改，该 Activity 即可使用多窗口模式，运行效果如图 4.55 所示。

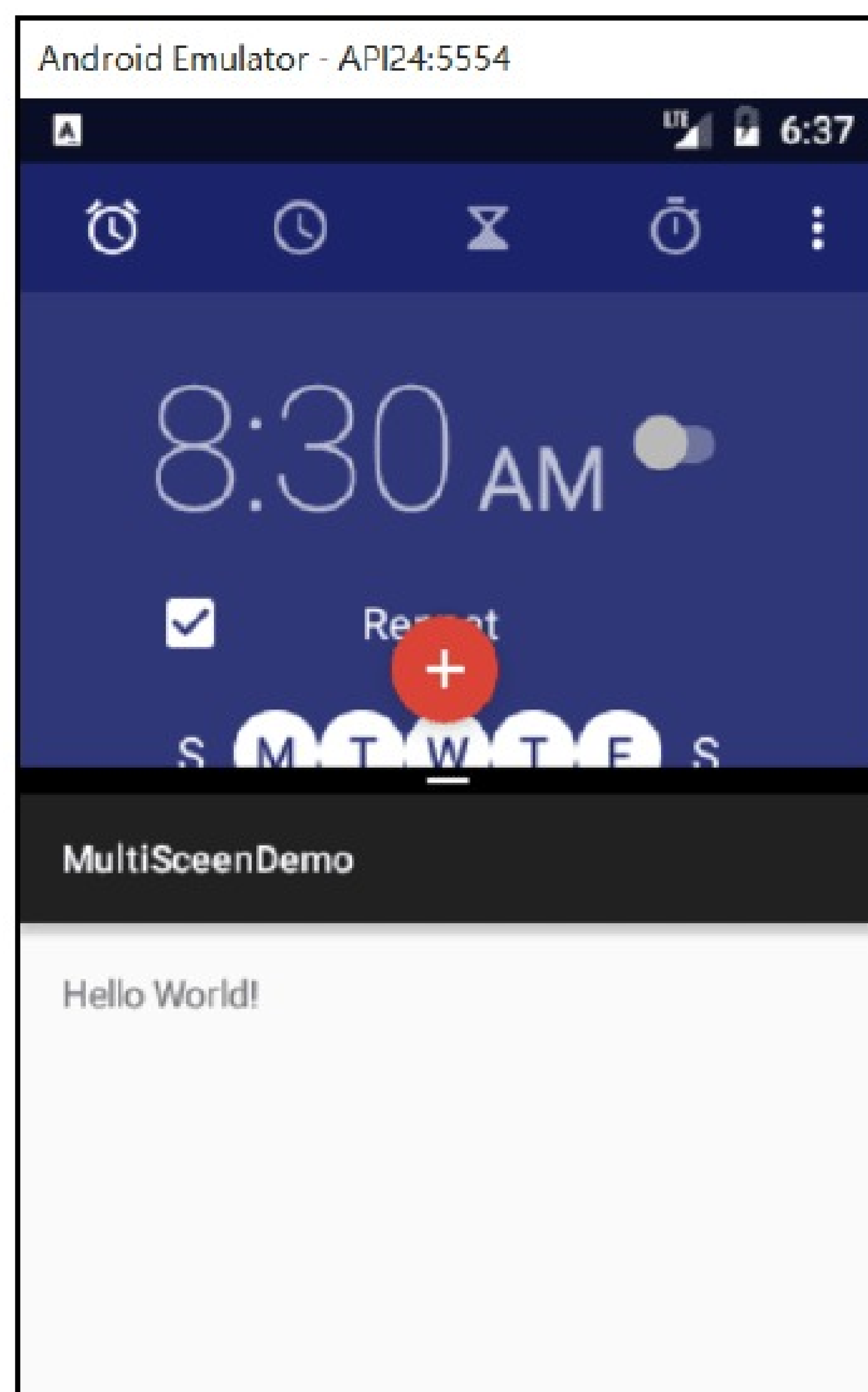


图 4.55 自开发应用的多窗口模式

我们在 MainActivity 上添加一个按钮，并实现点击打开第二个 Activity 的功能，代码如下：

```
public class MainActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button btn=findViewById(R.id.button);
        btn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Intent intent=new Intent(MainActivity.this,Main2Activity.class);
                startActivity(intent);
            }
        });
    }
```

点击“新窗口”按钮后，第二个窗口会被创建，并覆盖掉第一个窗口，如图 4.56 所示。

默认情况下，同一个应用的多个 Activity 会共用同一个窗口，且无法分配到不同窗口中。若希望同一个应用的不同窗体可以被分配到不同窗口中，需要在启动新窗体时给 Intent 设置一个 FLAG_ACTIVITY_LAUNCH_ADJACENT 标志，这样新 Activity 就会在新的栈中被启动，独立于原来的 Activity，进而实现两个 Activity 被放置于不同的窗口中，如图 4.57 所示。

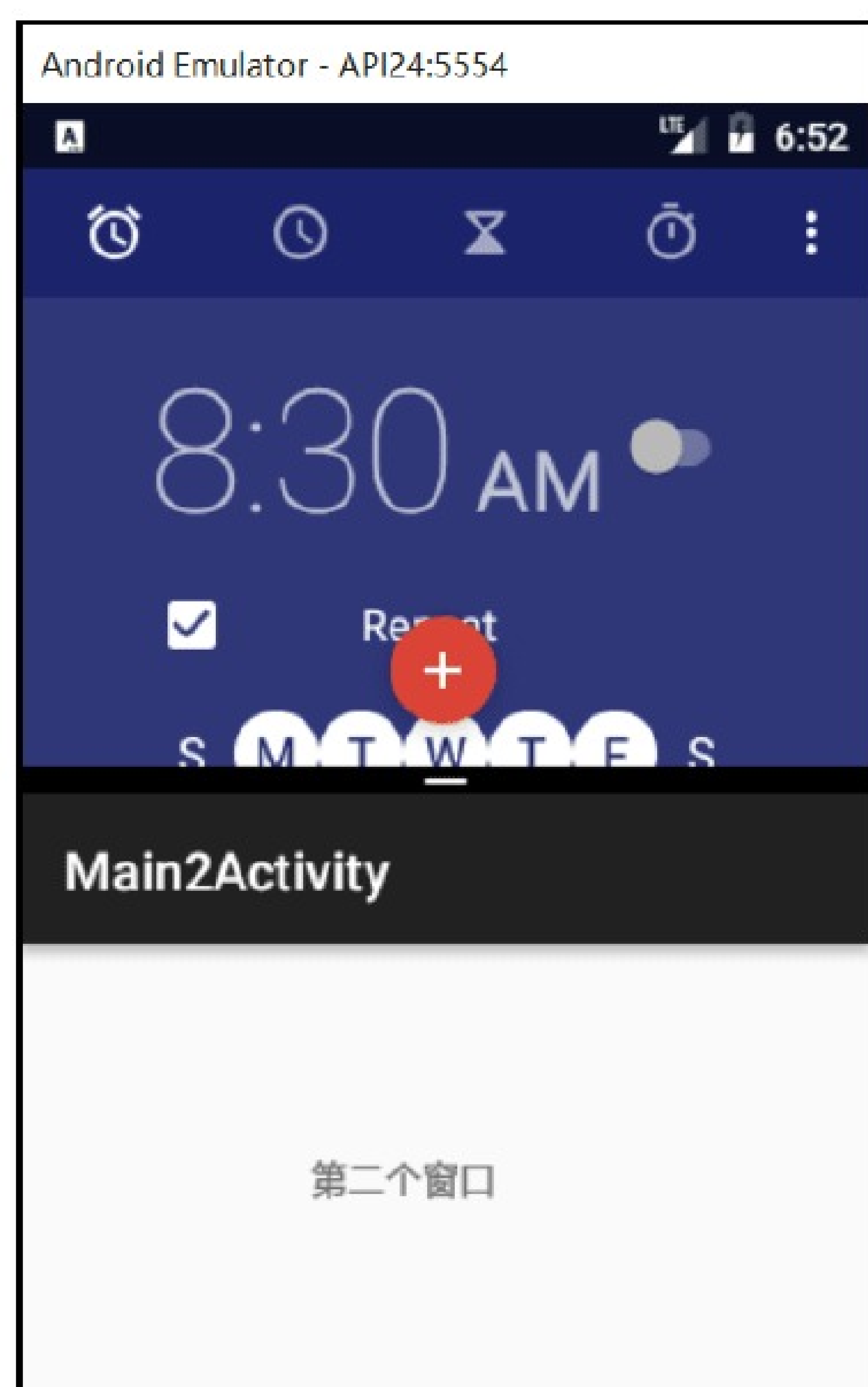


图 4.56　新窗口

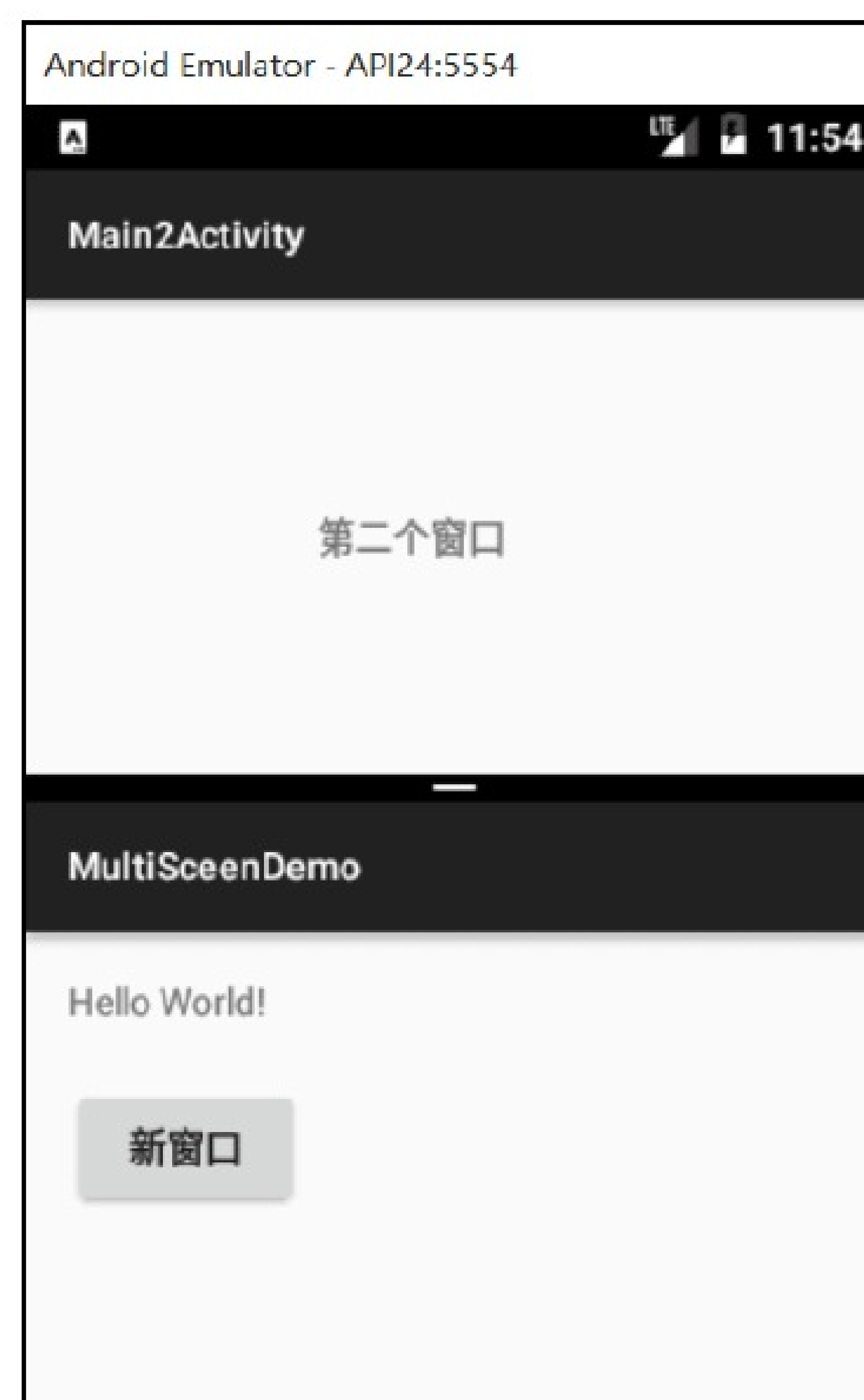


图 4.57　同一应用的两个窗口

关键代码如下：

```
    Intent intent=new Intent(MainActivity.this,Main2Activity.class);

intent.setFlags(Intent.FLAG_ACTIVITY_LAUNCH_ADJACENT|Intent.FLAG_ACTIVITY_NEW_TASK);
                startActivity(intent);
```

Android N 系统为 Activity 增添了<layout> 清单元素对 Activity 在多窗口模式中的行为进行支持，包括以下几种属性：

- android:defaultWidth，以自由形状模式启动时 Activity 的默认宽度。
- android:defaultHeight，以自由形状模式启动时 Activity 的默认高度。
- android:gravity，以自由形状模式启动时 Activity 的初始位置。请参阅 Gravity 的参考资料，了解合适的值进行设置。
- android:minimalHeight、android:minimalWidth，分屏和自由形状模式中 Activity 的最小高度和最小宽度。如果用户在分屏模式中移动分界线，使 Activity 尺寸低于指定的最小值，系统会将 Activity 裁剪为用户请求的尺寸。

例如，以下代码显示了如何指定 Activity 在自由形状模式显示时 Activity 的默认大小、位置和最小尺寸：

```
<activity android:name=".MyActivity">
    <layout android:defaultHeight="500dp"
          android:defaultWidth="600dp"
          android:gravity="top|end"
          android:minimalHeight="450dp"
          android:minimalWidth="300dp" />
</activity>
```

如果不想让 Activity 使用多窗口模式，只需要在清单文件中为 Activity 节点设置：

```
android:resizeableActivity="false"
```

此属性设置为 false，Activity 将不支持多窗口模式。在该值为 false 的情况下，如果用户尝试在多窗口模式下启动 Activity，该 Activity 将全屏显示。各位读者可以自行尝试。

4.10 界面事件响应

事件是 Android 平台与用户交互的手段。当用户对手机进行操作时，会产生各种各样的输入事件，Android 框架捕获到这些事件，进而进行处理。Android 平台提供了多种用于获取用户输入事件的方式，考虑到用户事件都是在特定的用户界面中产生的，因此 Android 选用特定 View 组件来获取用户输入事件的方式，由 View 组件提供事件的处理方法。这就是为什么 View 类内部带有处理特定事件的监听器。

4.10.1 事件监听器

监听器用于对特定事件进行监听，一旦监听到特定事件，则由监听器截获该事件，并回调自身的特定方法对事件进行处理。在本章之前的实例中，我们使用的事件处理方式都是监听器。根据用户输入方式的不同，View 组件将截获的事件分为 6 种，对应以下 6 种事件监听器接口。

（1）OnClickListener 接口：此接口处理的是单击事件，例如，在 View 上进行单击动作，在 View 获得焦点的情况下单击“确定”按钮或者单击轨迹球都会触发该事件。当单击事件发生时，OnClickListener 接口会回调 public void onClick（View v）方法对事件进行处理。其中参数 v 指的是发生单击事件的 View 组件。

（2）OnLongClickListener 接口：此接口处理的是长按事件，当长时间按住某个 View 组件时触发该事件。其对应的回调方法为 public boolean onLongClick（View v），当返回 true 时，表示已经处理完此事件，若事件未处理完，则返回 false，该事件还可以继续被其他监听器捕获并处理。

（3）OnFocusChangeListener 接口：此接口用于处理 View 组件焦点改变事件。当 View 组件失去或获得焦点时会触发该事件，其对应的回调方法为 public void onFocusChange（View v, Boolean hasFocus），其中参数 v 表示产生事件的事件源，hasFocus 表示事件源的状态，即是否获得焦点。

（4）OnKeyListener 接口：此接口用于对手机键盘事件进行监听，当 View 获得焦点并且键盘被敲击时会触发该事件。其对应的回调方法为 public boolean onKey（View v, int keyCode, KeyEvent event），其中参数 keyCode 为键盘码，参数 event 为键盘事件封装类的对象。

（5）OnTouchListener 接口：此接口用来处理手机屏幕事件，当在 View 的范围内有触摸、按下、抬起、滑动等动作时都会触发该事件，并触发该接口中的回调方法，其对应的回调方法为 public boolean onTouch（View v, MotionEvent event），对应的参数同上。

（6）OnCreateContextMenuListener 接口：此接口用于处理上下文菜单被创建的事件，其对应的回调方法为 public void onCreateContextMenu（ContextMenu menu, View v, ContextMenuInfo info），其中参数 menu 为事件的上下文菜单，参数 info 是该对象中封装了有关上下文菜单的其他信息。在

4.5 节的实例 MenusDemo 中，创建上下文菜单使用的是 registerForContextMenu（View v）方法，其本质是为 View 组件 v 注册该接口，并实现了相应的回调方法。

4.10.2 回调事件响应

在 Android 框架中，除了可以使用监听器进行事件处理之外，还可以通过回调机制进行事件处理。Android SDK 为 View 组件提供了 5 个默认的回调方法，如果某个事件没有被任意一个 View 处理，就会在 Activity 中调用响应的回调方法，这些方法分别说明如下。

（1）public boolean onKeyDown（int keyCode, KeyEvent event）方法是接口 KeyEvent. Callback 中的抽象方法，当键盘按键被按下时由系统调用。参数 keyCode 即键盘码，系统根据键盘码得知按下的是哪个按钮。参数 event 为按钮事件的对象，包含触发事件的详细信息，例如事件的类型、状态等。当此方法的返回值为 True 时，代表已完成处理此事件，返回 false 表示该事件还可以被其他监听器处理。

（2）public boolean onKeyUp（int keyCode, KeyEvent event）方法也是接口 KeyEvent. Callback 中的抽象方法，当按钮向上弹起时被调用，参数与 onKeyDown()完全相同。

（3）public boolean onTouchEvent（MotionEvent event）方法在 View 中定义，当用户触摸屏幕时被自动调用。参数 event 为触摸事件封装类的对象，封装了该事件的相关信息。当用户触摸到屏幕，屏幕被按下时，MotionEvent.getAction()的值为 MotionEvent.ACTION_ DOWN；当用户将触控物体离开屏幕时，MotionEvent.getAction()的值为 MotionEvent. ACTION_UP；当触控物体在屏幕上滑动时，MotionEvent.getAction()的值为 MotionEvent. ACTION_MOVE。onTouchEvent 方法的返回值为 true 表示事件处理完成，返回 false 表示未完成。

（4）public boolean onTrackballEvent（MotionEvent event）方法的功能是处理手机中轨迹球的相关事件，可以在 Activity 中重写，也可以在 View 中重写。参数 event 为手机轨迹球事件封装类的对象。该方法的返回值为 true 表示事件处理完成，返回值为 false 表示未完成。

（5）protected void onFocusChanged（boolean gainFocus, int direction, Rect previouslyFocusedRect）方法只能在 View 中重写，当 View 组件焦点改变时被自动调用，参数 gainFocus 表示触发该事件的 View 是否获得了焦点，获得焦点为 true，参数 direction 表示焦点移动的方向，参数 previouslyFocusedRect 是在触发事件的 View 的坐标系中前一个获得焦点的矩形区域。

4.10.3 界面事件响应实例

在之前的章节中，多次使用监听器对事件进行处理，读者应该已经很熟悉了。本节通过一个实例来演示回调事件响应的处理过程，该实例 EventDemo 的运行效果如图 4.58 所示。



图 4.58　实例 EventDemo 的运行效果

其布局文件 main.xml 内容如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical">

  <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="回调事件处理演示" />
<LinearLayout
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="horizontal">
  <Button
         android:id="@+id/button1"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:focusableInTouchMode="true"
         android:text="按钮1"/>
    <Button
         android:id="@+id/button2"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:focusableInTouchMode="true"
         android:text="按钮2"/>
    <Button
         android:id="@+id/button3"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
```

```
        android:focusableInTouchMode="true"
        android:text="按钮 3"/>
</LinearLayout>

</LinearLayout>
```

当用户在屏幕上做移动触摸、单击按钮等操作时，主 Activity EventDemo 会捕获相应事件并进行处理，在 LogCat 中打印相关内容，运行效果如图 4.59 所示。

Application	Tag	Text
introduction.android.eventDemo	enentDemo	第一个按钮获得了焦点
introduction.android.eventDemo	enentDemo	第二个按钮获得了焦点
introduction.android.eventDemo	enentDemo	手指正在往屏幕上按下
introduction.android.eventDemo	enentDemo	手指正在屏幕上移动
introduction.android.eventDemo	enentDemo	手指正在屏幕上移动
introduction.android.eventDemo	enentDemo	手指正在屏幕上移动
introduction.android.eventDemo	enentDemo	手指正在屏幕上移动
introduction.android.eventDemo	enentDemo	手指正在屏幕上移动
introduction.android.eventDemo	enentDemo	手指正在屏幕上移动
introduction.android.eventDemo	enentDemo	手指正在从屏幕上抬起

图 4.59　Activity EventDemo 捕获事件

EventDemo.java 代码如下：

```
package introduction.android.eventDemo;

import android.app.Activity;
import android.content.Context;
import android.graphics.Rect;
import android.os.Bundle;
import android.util.Log;
import android.view.KeyEvent;
import android.view.MotionEvent;
import android.view.View;
import android.view.View.OnFocusChangeListener;
import android.widget.Button;
import android.widget.Toast;

public class EventDemo extends Activity implements OnFocusChangeListener {
    Button[] buttons=new Button[3];
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        buttons[0]=(Button) findViewById(R.id.button1);
        buttons[1]=(Button) findViewById(R.id.button2);
        buttons[2]=(Button) findViewById(R.id.button3);
        for (Button button :buttons) {
            button.setOnFocusChangeListener(this);
        }
    }
    //按钮按下触发的事件
    public boolean onKeyDown(int keyCode,KeyEvent event)
```

```
    {
        switch (keyCode)
        {
            case KeyEvent.KEYCODE_DPAD_UP:
                DisplayInformation ("按下上方向键，KEYCODE_DPAD_UP") ;
                break;
            case KeyEvent.KEYCODE_DPAD_DOWN:
                DisplayInformation ("按下下方向键，KEYCODE_DPAD_UP") ;
                break;
        }
        return false;
    }
    //按钮弹起触发的事件
    public boolean onKeyUp (int keyCode,KeyEvent event)
    {
        switch (keyCode)
        {
            case KeyEvent.KEYCODE_DPAD_UP:
                DisplayInformation ("松开上方向键，KEYCODE_DPAD_UP") ;
                break;
            case KeyEvent.KEYCODE_DPAD_DOWN:
                DisplayInformation ("松开下方向键，KEYCODE_DPAD_UP") ;
                break;
        }
        return false;

    }

    //触摸事件
    public boolean onTouchEvent (MotionEvent event)
    {
        switch (event.getAction()) {
      case MotionEvent.ACTION_DOWN:
          DisplayInformation ("手指正在往屏幕上按下") ;
          break;
      case MotionEvent.ACTION_MOVE:
          DisplayInformation ("手指正在屏幕上移动") ;
          break;
      case MotionEvent.ACTION_UP:
          DisplayInformation ("手指正在从屏幕上抬起") ;
          break;
      }
        return false;
    }

    //焦点事件
    @Override
    public void onFocusChange (View view, boolean arg1) {
      switch (view.getId()) {
      case R.id.button1:
          DisplayInformation ("第一个按钮获得了焦点") ;
          break;
      case R.id.button2:
          DisplayInformation ("第二个按钮获得了焦点") ;
          break;
```

```
        case R.id.button3:
            DisplayInformation("第三个按钮获得了焦点");
            break;
        }
      }

      //显示 Toast
      public void DisplayInformation(String string)
      {
         //Toast.makeText(EventDemo.this,string,Toast.LENGTH_SHORT).show();
        Log.i("enentDemo",string);
      }
}
```

4.10 小　　结

本章介绍了使用 Android SDK 进行用户界面设计开发过程中涉及的相关知识，介绍了常用的 Widget 组件及布局的使用方法，菜单（Menu）和活动栏（ActionBar）结合使用的方法，AlertDialog 和 ProgressDialog 对话框的使用方法，Toast 和 Notification 的使用方法。每一部分都辅以实例，希望可以帮助读者掌握进行各个组件的使用方法，以后在 Android UI 的设计中能够有游刃有余。本章最后简单介绍了 Android 系统框架提供的组件与用户之间的交互事件的两种处理方法，分别为事件监听器和系统回调方法。

由于篇幅有限，在本章中不可能对用户界面设计中涉及的所有组件进行全面介绍，读者可以参阅 Android SDK 文档获取更多的知识。

4.11 习　　题

1. 为什么要使用布局？
2. 使用 TextView、EditText 和 Button 组件实现一个简单的计算器。
3. 实现一个输入密码框。在 EditText 中输入密码，当输入密码时显示“●”，在下方添加一个 CheckBox 用来选择是否“显示密码”。
4. 举例说明 CheckBox 与 RadioGroup 的区别有哪些。

第 5 章

电话和短信应用程序开发

手机的基本功能是打电话和发短信。本章通过 Intent 的使用来介绍在 Android 系统下如何对电话和短信应用程序进行开发。通过 Intent，程序员可以方便地将自己开发的应用程序与手机中的其他应用组件进行交互。

5.1　Intent

Intent 被译作“意图”，在 Android 中提供了 Intent 机制来协助应用间的交互与通信。Intent 负责对应用中一次操作的动作、动作涉及数据、附加数据进行描述，Android 则根据此 Intent 的描述，负责找到对应的组件，将 Intent 传递给调用的组件，并完成组件的调用。Intent 不仅可用于应用程序之间，也可用于应用程序内部 Activity/Service 之间的交互。因此，可以将 Intent 理解为不同组件之间通信的“媒介”，专门提供组件互相调用的相关信息。

Intent 是对它要完成的动作的一种抽象描述，Intent 封装了它要执行动作的属性：Action（动作）、Data（数据）、Category（类别）、Type（类型）、Component（组件信息）和 Extras（附加信息）。

1. Action

Action 是指 Intent 要实施的动作，是一个字符串常量。如果指明了一个 Action，执行者就会依照这个动作的指示，接收相关输入，表现对应行为，产生符合条件的输出。

在 Intent 类中定义了大量的 Action 常量属性，标准的 Activity Actions 如表 5.1 所示。

表 5.1　标准的 Activity Actions

动作名称	动作功能
ACTION_MAIN	作为一个主要的进入口，而并不期望去接收数据
ACTION_VIEW	向用户显示数据

（续表）

动作名称	动作功能
ACTION_ATTACH_DATA	用于指定一些数据应该附属于哪些地方，例如，图片数据应该附属于联系人
ACTION_EDIT	访问已给的数据，提供明确的可编辑接口
ACTION_PICK	从数据中选择一个子项目，并返回所选中的项目
ACTION_CHOOSER	显示一个 Activity 选择器，允许用户在进程之前选择他们想要的
ACTION_GET_CONTENT	允许用户选择特殊种类的数据，并返回（特殊种类的数据：照一张相片或录一段音）
ACTION_DIAL	拨打一个指定的号码，显示一个带有号码的用户界面，允许用户去启动呼叫
ACTION_CALL	根据指定的数据执行一次呼叫
ACTION_SEND	传递数据，被传送的数据没有指定
ACTION_SENDTO	发送一个信息到某个指定的人
ACTION_ANSWER	处理一个打进电话呼叫
ACTION_INSERT	插入一条空项目到已给的容器
ACTION_DELETE	从容器中删除已给的数据
ACTION_RUN	运行数据
ACTION_SYNC	同步执行一个数据
ACTION_PICK_ACTIVITY	为已知的 Intent 选择一个 Activity，返回被选中的类
ACTION_SEARCH	执行一次搜索
ACTION_WEB_SEARCH	执行一次 Web 搜索
ACTION_FACTORY_TEST	工厂测试的主要进入点

2. Data

Intent 的 Data 属性是执行动作的 URI 和 MIME 类型，不同的 Action 有不同的 Data 数据指定。例如，通讯录中 identifier 为 1 的联系人的信息（一般以 U 形式描述），给这个人打电话的语句为：

```
ACTION_VIEW  content://contacts/1
ACTION_DIAL  content://contacts/1
```

3. Category

Intent 中的 Category 属性起着对 Action 补充说明的作用。通过 Action，配合 Data 或 Type 可以准确表达出一个完整的意图（加一些约束会更精准）。Intent 中的 Category 属性用于执行 Action 的附加信息。例如，CATEGORY_LAUNCHER 表示加载程序时 Activity 出现在最上面，_HOME 表示回到 Home 界面。

4. Type

Intent 的 Type 属性显示指定 Intent 的数据类型（MIME）。通常 Intent 的数据类型可以从 Data 自身判断出来，但是一旦指定了 Type 类型，就会强制使用 Type 指定的类型而不再进行推导。

5. Component

Intent 的 Compotent 属性指定 Intent 的目标组件的类名称。通常情况下，Android 会根据 Intent 中包含的其他属性的信息进行查找，比如用 Action、Data、Type、Category 去描述一个请求，这种模式称为 Implicit Intents。通过这种模式，提供一种灵活可扩展的模式，给用户和第三方应用一个

选择权。例如，一个邮箱软件，大部分功能都不错，但是选择图片的功能不尽人意，如果采用 Implicit Intents，那么它就是一个开放的体系，如果想用手机中的其他图片代替邮箱中默认的图片，可以完成这一功能。但该模式需要付出性能上的开销，因为毕竟存在一个检索过程。于是 Android 提供了另一种模式 Explicit Intents ，该模式需要 Component 对象。Component 就是完整的类名，形如 com.xxxxx.xxxx ，一旦指明就可以直接调用。根据该属性是否被指定，Intent 可分为显式 Intent 和隐式 Intent。

6. Extra

Intent 的 Extra 属性用于添加一些组件的附加信息。比如，要通过一个 Activity 执行“发送电子邮件”这个动作请求，可以将电子邮件的 subject、body 等保存在 Extras 里，传给电子邮件发送组件。

5.1.1　显式 Intent 和隐式 Intent

Intent 寻找目标组件的方式分为两种：显式 Intent 和隐式 Intent。

显式 Intent 是通过指定 Intent 组件名称来实现的，它一般用在源组件已知目标组件名称的前提下，这种方式一般在应用程序内部实现。比如在某应用程序内，一个 Activity 启动一个 Service。

在不同应用程序之间，在不知道目标组件名称的情况下，寻找目标组件需要使用隐式 Intent。这种方式是通过 IntentFilter 实现的。

5.1.2　IntentFilter

为了支持隐式 Intent，可以声明一个甚至多个 IntentFilter。每个 IntentFilter 描述组件所能响应 Intent 请求的能力。比如请求网页浏览器，网页浏览器程序的 IntentFliter 就应该声明它所希望接收的 IntentFilter Action 是 WEB_SEARCH_ACTION，以及与之相关的请求数据是网页地址 URI 格式。

如何为组件声明自己的 IntentFilter？常见的方法是在 Android Manifest.xml 文件中用属性 <Intent-Filter>描述组件的 IntentFilter。

一个隐式 Intent 请求要能够传递目标组件，必要通过 Action、Data 以及 Category 三方面的检查。任何一方面不匹配，Android 都不会将该隐式 Intent 传递给目标组件。接下来我们讲解这三方面检查的具体规则。

1. 动作测试

<intent-Filter>元素中可以包含子元素<action>，比如：

```
<intent-Filter>
  <action android:name=“com.example.project.SHOW-CURRENT” />
<action android:name=“com.example.project.SHOW-RECENT” />
<action android:name=“com.example.project.SHOW-PENDING” />
</intent-Filter>
```

一条<intent-Filter>元素至少包含一个<action>，否则任何 Intent 请求都不能和该<intent-Filter>匹配。